DTIC FILE COPY

AD-A228 470

IBM STARS REPOSITORY GUIDEBOOK

April 30, 1990

DTIC
ELECTE
NOV 09 1990
S B D

Contract No. F19628-88-D-0032
Task IR40: Repository Integration

Delivered as part of:
CDRL Sequence No. 1550

Prepared for:

Electronic Systems Division
Air Force Systems Command, USAF
Hanscomb AFB, MA 01731-5000

Prepared by:

IBM Systems Integration Division
800 North Frederick
Gaithersburg, MD 20879

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

90 11 3 123

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (*Leave blank*)

2. REPORT DATE
April 30, 1990

3. REPORT TYPE AND DATES COVERED
Final

4. TITLE AND SUBTITLE
[redacted]
See Below

5. FUNDING NUMBERS
C: F19628-88-D-0032

6. AUTHOR(S)
R. Ekman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
IBM Federal Sector Division
800 N. Frederick Avenue
Gaithersburg, MD 20879

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
CDRL Sequence No. 1550

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

A guide to software reuse using the STARS Repository. This document contains the IBM STARS Repository Guidebook, STARS Repository User's Guide, and STARS Reusability Guidelines. Each is described below.

IBM STARS Repository Guidebook. A guide to the STARS Repository, providing high-level information for all users -- component reusers, component suppliers, and repository administrators. The Guidebook is organized according to the specific roles that users perform when using the system.

STARS Repository User's Guide. A guide on how to access and use the STARS Repository. It provides the basic information needed to use the repository software, but it is not a comprehensive guide to the VAX computer, on which the repository is built.

STARS Reusability Guidelines. A set of Ada coding guidelines for component development that emphasize reusability. Code that follows these guidelines will be easier to reuse on multiple projects and platforms. Many examples are provided illustrating the guidelines.

14. SUBJECT TERMS
STARS, software reuse, software reuse library, Ada coding guidelines, Ada

15. NUMBER OF PAGES
201

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

(Software Technology for Adaptable Reliable Systems)

Abstract

This Guidebook provides high-level information for all users of the IBM STARS Repository. It describes how to use the Repository system in the context of software reuse.

The Guidebook is organized according to specific roles that users perform when using the system. It addresses the information needs of component reusers, component suppliers, and repository administrators.

This Guidebook defines:

- 1) How to use the IBM STARS Repository,
- 2) Resources and capabilities of the Repository,
- 3) Component standards for admission to and promotion within the Repository,
- 4) Procedures for submission and usage of repository components,
- 5) Processes involved in managing repository assets, and
- 6) The reuse process in the context of the STARS repository.

The Guidebook is the key document in a suite of Repository guides. The other documents are the *STARS Reuseability Guidelines*[IBMCode] and the *IBM STARS Repository User's Guide*[IBMUser]. These documents were separated from the Guidebook for practical reasons, but they are required to make full use of the Guidebook.

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By per letter	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Preface

The content of the Guidebook is tightly coupled to the IBM STARS Repository. It will be reviewed periodically, incorporating the lessons learned from using the guidelines and procedures, and updating as the Repository matures. In the short term, the Guidebook will help meet the practical needs of the IBM STARS development teams. In the long term, it will serve as a baseline element in a more automated software development environment.

The guidelines and procedures in the Guidebook apply to all IBM STARS team tasks. In particular, they apply to Ada code and related documentation as it is admitted to and managed within the Repository. The Guidebook serves as a starting point for development in future increments of the STARS project.

The term 'Repository' has developed several meanings within the software development industry. In the STARS project, the term has been associated with a machine, a facility, and an application. In general, this Guidebook uses the latter meaning, also referring to the Repository as a *Reuse Library*.

This Guidebook was developed by the IBM Systems Integration Division, located at 800 North Frederick, Gaithersburg, MD 20879. Questions or comments should be directed to the author, Robert W. Ekman at (301) 240-6431, or to the IBM STARS Program Office.

Table of Contents

Introduction

Welcome to the IBM STARS Repository.

The IBM STARS Repository facilitates the distribution and sharing of software, documentation, and project information within the STARS community [RFP87]. It also serves as a bridge between the STARS community and the software engineering industry.

In particular, the Repository

- stores and organizes designated material,
- provides capabilities for convenient searching and retrieval,
- provides capabilities for electronic dialogue (e.g., mail, conferencing, etc.) among its user community, and
- provides 24-hour access for convenient delivery or copying of material over government or public electronic networks.

In addition, the Repository

- is an instance of the STARS software engineering environment, and
- presents a vision of the software reuse process model.

The Repository supports

- software engineering through the software reuse model,
- project coordination through work product sharing and prototyping, and
- contract administration by archiving deliveries.

This Guidebook is your introduction to the IBM STARS Repository. It gathers in one place all the references, guidelines, and procedures related to the operation and use of the Repository [IBM1540]. It will give you an understanding of the nature and scope of the Repository.

After skimming this guidebook, you should use the Repository system. Only through hands-on exercise of the system will you become comfortable with it. A section in the Appendix of this document describes how you can get an account and access the system.

The Repository is a maturing system with a component supply and problem correction process. You will notice improvements in the Repository over what is described in this Guidebook. The component content of the reuse library is dynamic. New components are added every week and existing components are being reviewed.

You are encouraged to make the most out of the system and its features, and to report problems and suggestions. The Repository support team is available and willing to help.

Reuse and the Repository

A repository is the primary support tool for software reuse and is a key element in project life cycle integration [SEI89]. Without some form of a repository or reuse library, significant reuse on a software development project is impossible. Reuse starts during early phases of the project [RSL87]. You must set goals for reuse and commit to not "reinvent the wheel".

Your project's problem definition and solution is related to a particular domain or set of domains, such as 'software engineering' or 'air traffic control'. Likewise, repositories are related to specific domains. To be successful with reuse, you must locate repositories that support the domains related to your project. You must become familiar with the repository access and content as early as possible.

The system engineers must define architectures with a view toward reuse. Requirements and specifications must be defined recognizing the availability and capability of reusable components. Reuse must be specified. If you start building without reuse specifications, you cannot take advantage of the significant quality and cost benefits of software reuse.

During implementation you bring together the components from the repository. Large grain components are most desirable. They will form the major parts of the solution. Small grain components are incorporated while editing and developing new system elements. During new code creation, you should keep open access to the repository. This will allow you to interrogate the repository for candidate components as you develop code.

As you complete your project or incremental release, you should consider putting key components from your development into the repository. If you made modifications to repository components, then you should return them as new versions. The reuse-repository model is a cyclic process, with supply and consumption the driving forces.

The basic model of the Repository system consists of three user environments: the repository platform, the software engineering environment, and the external casual user system. The repository platform consists of two major elements: the Repository application and the user communication mechanisms. The software engineering environment consists of a collection of tools coordinated through an object manager. The external user system provides access to the repository elements from outside the software engineering team. These elements and their relationships are presented in Figure 1.

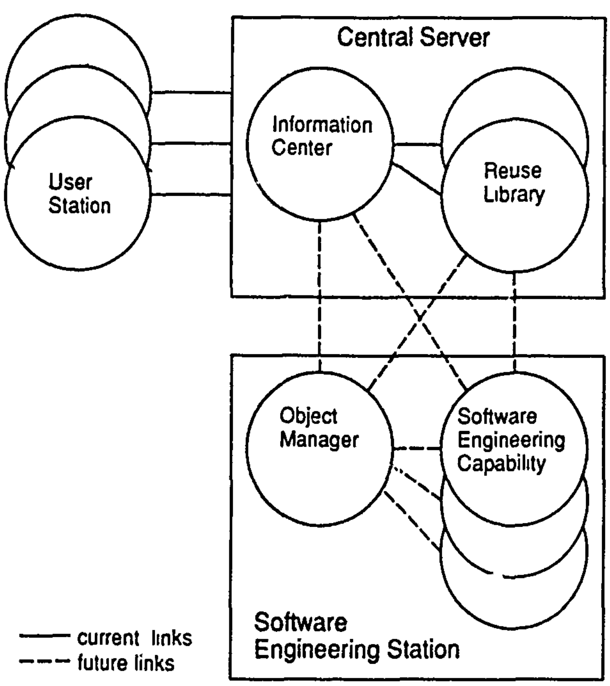


Figure 1: Basic Repository System Model

Definitions of Key Terms

In order to understand the processes described in the Guidebook and represented in the Repository implementation, you should review the following terms and definitions. They will help you relate this system to your previous experiences in libraries and software development. They are in an order that permits sequential reading, rather than an alphabetic order. Additional terms used in the Guidebook and the Repository system are defined in "Glossary" on page 10.

component A collection of related work products to be used as a consistent set of information. Software work products can include specifications, design, source code, machine code, reports, compilation units, code fragments and other components. It is the primary object type that is the concern of a repository. Also referred to as an **asset**, or **resource**.

part An element of a component, such as design documents, source code, test information, and data rights. While a part may be copied or browsed, when stored in a repository it is always associated with a component.

repository An element of the software engineering environment in which software work products and information about them are stored. Its primary purpose is support of reuse. Also referred to as a **reuse library**.

repository system An instance of the software engineering environment, including computer hardware, operating systems, software tools, standards, and procedures that together provide a complete repository capability. These capabilities include acquiring, storing, managing, retrieving, and dispensing software work products and information about them. A repository system may contain several repositories, each dealing with a different problem domain.

depository A repository or part of a repository whose contents are deposited as is, with few or no constraints. A depository is primarily a storage facility. Quality and usability are unpredictable; the burden is on the user to find and evaluate useful items.

organized repository A repository whose contents are documented and organized in a comprehensive manner as components and parts. Finding, reviewing, and extracting components are supported. The quality and usability remain unpredictable.

filtered repository A repository whose components are subjected to standards on form, content, quality, and consistency. This is not a physical partition from the organized repository, rather it is a documented higher level of confidence in an existing organized component.

certified repository A repository whose components exhibit the highest level of confidence. A certified component permits the proving of correctness in a solution that reuses the component. Aspects of security, ownership, distribution, and user set take on greater importance.

reuser A person who reviews the contents of a repository and extracts components for reuse. The common user of a repository. Also referred to as a **user**.

supplier A person places components into a repository. Also referred to as a **contributor**, or **submitter**.

librarian A person who coordinates, organizes, and controls the contents of a repository. Also referred to as a **system administrator**, **database administrator**, or **repository administrator**. This person frequently provides a first level help for other users.

topic specialist A person who is responsible for the evaluation of components and promoting them to the filtered of certified status. This person understands the repository domain and has reviewed many of the components in the repository. Also referred to as a **domain expert**, or **technical consultant**. This person frequently provides a second level help for other users, via reference from the librarian.

component data file A file containing structured information about the component. The file is prepared automatically, based on answers to questions in the on-line supply process. It is the representation of the component when it is held in the librarian's 'holding bin'. It is reviewed by the librarian and facilitates the entry of the component into the repository. This file also forms the basis for component interchange with other repositories.

filtering The act of evaluating components and promoting them to filtered status.

certifying The act of certifying components and promoting them to certified status.

gatekeeper An automated capability that facilitates component evaluation. It is usually associated with entry to the repository and filtering, but may be used by suppliers to review their components before submitting to the repository, or as support for the certification process.

Repository User Roles

In using the Repository facilities and services, you and other individuals acting in various roles will interact to accomplish reuse. This guidebook is organized on the basis of these roles. For each role, references are made to the guidelines and procedures to be used in conducting your role. Each role carries with it duties and responsibilities that combine to make the repository mission a success.

It is important to realize that the role is distinct from the individual or individuals who perform in that role. It is possible, for instance, that a given role may be performed in whole or in part by a single individual, by more than one individual, or by several individuals working together. Individuals often perform several different roles when using the Repository.

Component Reuser

A *reuser* is a person who reviews the contents of a repository and extracts components for reuse within their development project. The satisfaction of the reuser is the primary focal point of the IBM STARS Repository. Through a large component supplies and effective retrieval mechanism the Repository will increase the user's degree of reuse.

Reuse is a simple, easy to accomplish process, but requires a confidence and commitment on the part of the reuser. First, you establish a general requirement for some form of reusable component. You express this need in general textual terms, such as 'window system', 'list processor', 'memory manager', or 'file browser'.

The next step is to access the Repository and invoke the component search facility. Using the menu interface, you construct a query that is broad enough to encompass many candidate components, but restrictive enough to eliminate obvious non-candidates. You then invoke the query, review the returned candidate list, and select specific components for extraction. You may want to browse the components from the candidate list to review the component before extraction.

In the extraction process, you will create copies of the selected components in your local directory on the Repository system. You then copy them to your development system. Once in your development system, you may compile the components and test them.

You may locate several components that meet your needs. You should consider extracting all of them and attempt to build a set of alternative solutions. In doing assessment of the alternatives you will develop a better understanding of your problem domain and will gain material for comparative analysis.

In using the Repository, the reuser takes on some responsibility. The reuser must follow the code of conduct as defined by the Repository administration [IBM1460, IBM1470]. Reusers should not redistribute components outside their project. If other people or projects want some of the Repository content, they must establish their own contact into the Repository. When reusers discover errors with a component or has problems with the Repository system, they should report them through the on-line problem reporting mechanisms.

Component Supplier

A *supplier* is a person who contributes components to the Repository. All reusers are potential suppliers, but a supplier is a reuser with a component that could be reused by another developer. The supplier provides feedback into the reuse process by first extracting and subscribing to components, and then by improving the component or developing new components and submitting them to the Repository

The supplier should be the author of the component, because the author will know the component attributes and be able to fill out the supply forms accurately. The supplier will become the point of contact for the component. Third party suppliers are not encouraged.

While it sounds simple, the supply process is complicated by data rights, ownership, incentives to supply, and amount of effort to supply. The effort to supply is inversely proportional to the reuse effort, and according to the economics of the process, it should be more difficult to supply than reuse. The IBM STARS Repository has not resolved these issues, but has provided a rudimentary facility in which to prototype the operational aspects of the issues.

The role of the supplier involves packaging and presenting work products for inclusion in the STARS Repository. The work products should adhere to the STARS guidelines for reusability and portability. You should pay particular attention to the issues of adaptation and tailoring of your components. If you are developing software and want more information on the STARS coding standards, refer to the *STARS Reusability Guidelines*[IBMCode]. You may also want to run your code through the an Ada analysis tool, such as AdaMAT which is available on the IBM STARS machine. The issues of coding standards should not deter you from submitting a component. The evaluation process of filtering will review the component for style and standards compliance. If you believe your component is reusable, then you should submit it.

As a supplier, you are responsible for ensuring that the component information requirements are satisfied prior to submitting work products to the Repository. This usually involves collection of detailed information about the component to be supplied. You will have to transfer the component parts to the Repository system. You then invoke the component supply facility and follow the on-line directions.

You may want to submit a small test component using the on-line supply facility first. You should take screen prints of the supply questions so that you can compile the required information off-line before you start with a 'real' component. The Repository Librarian will recognize your 'test' component and not place it in the Repository.

Developers in the STARS project form a special class of suppliers. All STARS developers should plan on becoming suppliers sometime during their development cycle. In adhering to the STARS guidelines and contract requirements, the STARS developers will produce products that are inherently ready for supply.

Repository Administration

Repository Manager

The *repository manager* owns and operates the Repository. The manager is responsible for establishing and maintaining Repository procedures and policies. The manager has overall responsibility for the Repository Librarian and Topic Specialists.

The manager has the following specific responsibilities:

- Maintain system availability
- Manage and review access control
- Store STARS contract delivery items
- Conduct system backup
- Implement disaster recovery and vital records plans

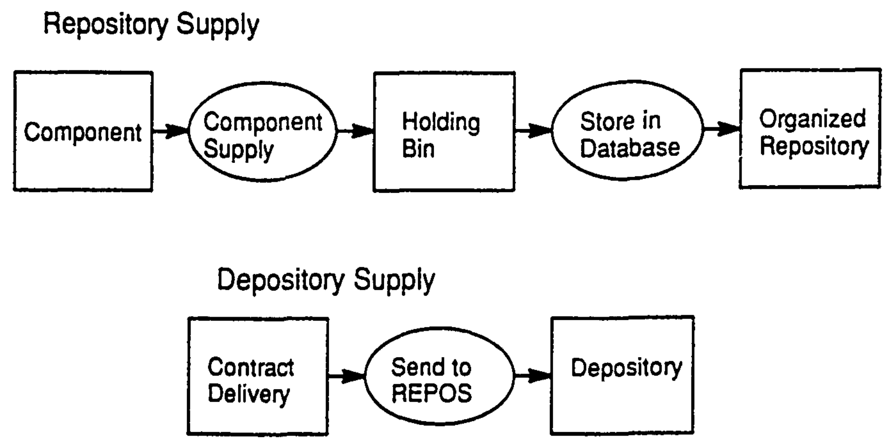
Repository Librarian

The *repository librarian* coordinates, organizes, and controls the contents of the Repository. The librarian acts as a database administrator. The librarian is available for help with reuse and supply problems.

The librarian has two different paths to follow, depending on the type of item submitted to the system. Figure 2 is a graphic illustration of these paths.

- Review submitted reusable material in the 'holding bin' and place the material in the organized repository, or

  Copy STARS contract deliverables to the depository with minimum inspection.



Fig

Figure 2: Repository Librarian Activities

When a component is placed in the 'holding bin', it is represented by a component data file. This file is reviewed by the librarian and may modify the file to bring the component into compliance with entry criteria. This form of the component is also used when exchanging components with other repositories.

Topic Specialist

The *topic specialists* review and evaluate components. They understand the repository domain and have reviewed many of the components in the repository. They are responsible for the evaluation of components and promoting them to filtered or certified status. The topic specialist frequently provides a second level help for other users, via reference from the librarian.

The act of evaluating and promoting components is referred to as 'filtering'. Specific filtering guidance is presented in the Appendix. The specialist uses an automated evaluation facility called the 'gatekeeper'. This facility is also available, as a prototype, for component developers and the librarian to evaluate components within their part of the reuse process.

The specialist role supports quality assurance of work products that are received into the repository by ensuring that the Repository reusability guidelines and portability guidelines are met. In addition, the specialist contributes to Repository configuration management by assuring that the work product is properly classified and properly inserted into the Repository structure.

References

There are many exciting successful reuse efforts completed or in-progress. All the major DoD software agencies and contractors are involved. Of particular interest are the McDonnell Douglas CAMP experiences [CAMP86] and the SEI Software Reuse Project [SEI89]. A literature search in your project's problem domain will turn up references appropriate for your project.

The following is a list of references in this Guidebook and its appendices in the order that they are cited. References that are exclusively used in the reusability coding guidelines are contained in that section only. The IBM STARS documents with reference tags of [IBMxxx] are available in electronic form from the depository.

[IBMCode] IBM Systems Integration Division, *STARS Reusability Guidelines*, April 30, 1990.

[IBMUser] IBM Systems Integration Division, *IBM STARS Repository User's Guide*, April 30, 1990.

[RFP87] United States Department of Defense, Department of the Air Force, *STARS Competing Primes Lead Contracts Request For Proposal*, F19628-88-R-0011, November 5, 1987.

[IBM1540] IBM Systems Integration Division, *Repository Guidebook (Draft)*, CDRL Sequence No. 1540, September 14, 1989.

[SEI89] Software Engineering Institute, "Reuse: Where to Begin and Why," *Affiliates Symposium*, May 2-4, 1989.

[RSL87] Burton, B. A., and others, "The Reusable Software Library," *IEEE Software*, July 1987.

[IBM1460] IBM Systems Integration Division, *Draft Policies and Procedures*, CDRL Sequence No. 1460, January 19, 1990.

[IBM1470] IBM Systems Integration Division, *Repository Operations and Procedures*, CDRL Sequence No. 1470, March 7, 1990.

[CAMP89] McDonnell Douglas Astronautics Company, "Overview and Commonality Study Results,", *Common Ada Missile Packages (CAMP)*, AFATL-TR-85-93, May 1986.

[Peterson79] Peterson, A. S., "Coming to Terms with Terminology for Software Reuse," *Reuse in Practice Workshop*, 1989.

[IEEE729] IEEE, *Standard Glossary of Software Engineering Terminology*, ANSI/IEEE STD 729-1983, IEEE Standards Board, September 23, 1983.

[Webster88] Merriam-Webster Inc., *Webster's Ninth New Collegiate Dictionary*, Springfield Mass., 1988.

[IBM1440] IBM Systems Integration Division, *Practical Aspects of Repository Operations*, CDRL Sequence No. 1440, January 10, 1990.

[IBM380]	IBM Systems Integration Division, *Consolidated Reusability Guidelines*, CDRL Sequence No. 0380, March 21, 1989.
[IBM70]	IBM Systems Integration Division, *Consolidated Technical Development Plan for STARS Competing Prime Contractors*, CDRL Sequence No. 0070, November 11, 1989.
[IBM110]	IBM Systems Integration Division, *Environment Capability Matrix*, CDRL Sequence No. 0110, March 17, 1989.
[IBM1580]	IBM Systems Integration Division, *Taxonomy Report*, CDRL Sequence No. 1580, January 19, 1990.
[IBM1320]	IBM Systems Integration Division, *Quality Assurance/Configuration Management Plan*, CDRL Sequence No. 1320, October 20, 1989.
[IBM1600]	IBM Systems Integration Division, *Version Description Document for the IBM STARS Repository*, CDRL Sequence No. 1600, January 31, 1990.
[CUA89]	IBM, *Common User Access Advanced Interface Design Guide*, SC23-4582-0, June 1989.

Glossary

The terms and definitions used in the IBM STARS Repository are listed in "Definitions of Key Terms" on page 3. The following terms and definitions provide additional help in describing software reuse. They are extracted from the "Partial Glossary for Software Reuse", contained in a paper by A. Spencer Peterson (SEI) titled "Coming to Terms with Terminology for Software Reuse" [Peterson89].

The terms and definitions are taken from a draft update to ANSI/IEEE Std 729 (Glossary of SW Terminology) [IEEE729], except where the term is marked with an (M) for Modified where inserted text is enclosed in [ ], or with a (*) signifying a term that is not defined in the IEEE draft. Other comments are enclosed by { } and placed at the end of the definition.

abstract data type: A data type for which only the properties of the data and the operations to be performed on the data are specified, without concern for how the data will be represented or how the operations will be implemented.

abstraction: (1) A view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information. (2) The process of formulating a view as in (1).

adaptation data(M): Data used to adapt a program [or component] to a given installation site or to given conditions in its operational environment.

adaptation parameter(M): A variable [or placeholder] that is given a value [or other appropriate information] to adapt a program [or component] to a given installation site or to given conditions in its operational environment.

adaptive engineering(*): The process of modifying a system or component to perform its functions in a different manner or on different data than was originally intended.

adaptive maintenance(M): Software maintenance performed to make a computer program [or component] usable in a changed environment.

application-oriented language: A computer language with facilities or notations applicable primarily to a single application area.

architecture: The organizational structure of a system or component.

artifact(*): Any product of the software development process.

asset(*): a set of reusable resources that are related by virtue of being the inputs to various stages of the software life-cycle, including requirements, design, code, test cases, documentation, etc. {An asset can be design and the control code for using other assets in the library in a more powerful way. Assets are the fundamental element in a reusable software library.}

component: One of the parts that make up a system. {A component is some useful portion of a computer program. It may be subdivided into other components.}

control abstraction(*): (1) The process of extracting the essential characteristics of control by defining abstract mechanisms and their associated characteristics while disregarding low-level details and the entities to be controlled. (2) The result of the process in (1).

data abstraction: (1) The process of extracting the essential characteristics of data by defining data types and their associated functional characteristics and disregarding representational details. (2) The result of the process in (1).

domain(*): The set of current and future systems/subsystems marked by a set of common capabilities and data.

domain analysis(*): (1) The process of identifying, collecting, organizing, analyzing, and representing a domain model and software architecture from the study of existing systems, underlying theory, emerging technology, and development histories within the domain of interest. (2) The result of the process in (1).

domain engineering(*): The construction of components, methods, and tools and their supporting documentation to solve the problems of system/subsystem development by the application of the knowledge in the domain model and software architecture.

domain model(*): A definition of the functions, objects, data, and relationships in a domain.

functional abstraction(*): (1) The process of extracting the essential characteristics of desired functionality by defining its abstractly along with its associated behavioral characteristics and disregarding low-level details. (2) The result of the process in (1).

independence(*): The ability of a component to be used with different compilers, operating systems, machines and applications than those for which it was originally developed. Independence is closely related to *portability*.

maintainability(*): The ease of modifying a component, whether it be to meet particular needs or to fix bugs.

master library: A software library containing master copies of software and documentation from which working copies can be made for distribution and use. {This should be meticulously maintained and controlled by a special group of reuse engineers and librarians.}

modularity(M): The degree to which a system, computer program [or code component] is composed of discrete components such that a change to one component has minimal impact on other components.

perfective maintenance(M): Software maintenance performed to improve the performance, maintainability, or other attributes of a computer program [or component].

platform(*): Platform refers to the architecture for the system for which the product is intended (hardware, operating system, and Ada compiler). Some products may be intended for several different platforms. Platforms listed should also indicate whether they are host platforms, target platforms, or both.

portability(*): The ability of an application or component to be used again in a different target environment than the one it was originally built for. The phrase *target environment* may be defined broadly to include operating systems, machines, and applications. To be ported effectively, components may need to be tailored to the requirements of the new target environment. See also *reusability* and *independence*.

production library: A software library containing software approved for current operational use.

reliability(*): The extent to which a component performs as specified. A reusable component performs consistently with repeated use and across environments (that is, operating systems and hardware).

resource(*): Any software entity placed into a software library for purposes of reuse.

retirement(M): (1) Permanent removal of a system, component [or resource] from its operational environment [or the master library.] (2) Removal of support from a operational system, component, [or resource].

reusability(M): The degree to which [a] software [resource] can be used in more than one computer program [or system, or in building other components or parts.] {See also *portability*.}

reusable software(*): Software designed and implemented for the specific purpose of being reused.

reuse(*): The application of existing solutions to the problems of systems development.

reuse engineering(*): (1) The application of a disciplined, systematic, quantifiable approach to the development, operation and maintenance of software where reuse is a primary consideration in the approach. (2) The study of approaches as in (1). {The same definition as for 'software engineering' given in the IEEE standard except for the addition of the phrase beginning with 'where'.}

software(M): Computer programs, [code components and other artifacts], procedures, and possibly associated documentation and data pertaining to the operation of a computer system [or its components].

software architecture(*): The packaging of functions and objects, their interfaces, and control to implement applications in a domain.

software library(M): A controlled collection of software [resources] and related documentation designed to aid in software development, use, [reuse], or maintenance.

software repository: A software library providing permanent, archival storage for software and related documentation. {The key word is 'archival'. Also note the word 'control' is not mentioned.}

software reuse(*): (1) The process of implementing new software systems and components from pre-existing software. (2) The results of the process in (1).

specification(M): A document [or other media] that specifies, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or component, and, often, the procedures for determining whether or not these provisions have been satisfied.

tailorability(*): The ease of modifying a component to meet particular needs. It should be distinguished from *maintainability*, which includes tailorability, but also includes the idea of corrective maintenance (fixing bugs).

taxon(*): A group of resources constituting one of the categories in a taxonometric classification for reusable software in one or more domains. {The plural is taxa.} {A taxonomic group or entity [Webster88].}

taxonomy: The study of the general principles of scientific classification. [Webster88].

Acronyms

The following is a list of acronym, abbreviation, and similar terms used in this Guidebook and its appendices.

Acronym	*Meaning*
AdaMAT	an Ada Metric Analysis Tool by Dynamics Research Corp.
ADT	abstract data type
ANSI	American National Standards Institute
CDRL	Contract Data Requirements List
CUA	Common User Access
DEC	Digital Equipment Corporation
DoD	(United States) Department of Defense
DOS	Disk Operating System (for a personal computer)
DRC	Dynamics Research Corporation
IBM	International Business Machines
IEEE	Institute of Electrical and Electronic Engineers
IR40	IBM STARS R-increment task for Repository Integration
Oracle	a commercial relational database product
RFP	Request for Proposal
SAA	System Application Architecture
SAIC	Science Applications International Corporation
SEI	Software Engineering Institute
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
STARS	Software Technology for Adaptable, Reliable Systems
VAX	a computer system from Digital Equipment Corporation
VMS	a proprietary operating system for a VAX

Appendix A. Reusing Components

Repository Access

The IBM STARS Repository is accessed remotely from your system via dial-up or Internet connections. Complete user information is found in the *IBM STARS Repository User's Guide* [IBMUser]. This guide contains sections on

- Getting an account,
- Remote user system requirements,
- Making the connection, and
- System commands.

The policies and procedures governing your use of the system are documented in the Repository Policies and Procedures [IBM1460] and the Repository Operations and Procedures [IBM1470] documents. Additional information about the Repository System can be found in *Repository Operations* [IBM1440].

Repository Menu Application

The Repository application uses a menuing system that is based on a rudimentary window manager. To display the primary menu, enter "repos" at the system command prompt.

```
IBM STARS Team Repository
```

```
Component Search
Directory Search
Component Supply
Browse Current Catalog
Repository Tools
Repository Services
Suggestion Box
Problem Reporting
Help
Exit
```

Figure A-1: Primary Selection Menu

Each menu is composed of several options including an "Exit" option. Each option may be selected by moving the highlight bar to the option of interest (using the up and down arrow keys) and pressing Enter. A second means of selecting an option is to simply enter the highlighted letter in the option of interest.

Several of the options offer submenus. Return to the next higher level of the menu system is always accomplished by pressing "x" or moving the highlight bar to "Exit" and pressing Enter.

```
IBM STARS Team Repository

Compone
Directo   Repository Tools
Compone
Reposit
Reposit   Ada Dev
Suggest   Run SGM   Ada Development Tools
Problem   File Br
Help      Add A P
Exit      Add An    AdaMAT
          Problem   Edit Ada Source Code
          Help      Format Ada Source Code
          Exit      Check Ada Style
                    Count Ada Statements
                    Profile Ada Statements
                    Generate Compile Order
                    Compile Ada Programs
                    Problem Reporting
                    Help
                    Exit
```

Figure A-2: Example of Submenus

The Repository application has context sensitive help information. By pressing "h" a full screen of information will be displayed describing the capabilities of the current menu.

Component Reuser Procedures

The primary Repository capability used by the reuser will be the component search facilities. At first, these facilities will seem complicated, but after a few passes through the process you will appreciate the interface and be able to locate components of interest.

There are two basic search facilities:

- Component Search - direct retrieval based on component attributes. Within the database there are two primary types of components: software and document.
- Directory Search - selection of depository directory and then files, ordered by project and tasks.

The following figure presents the user interface search fields for the software attributes in the database search facility.

```
  F6=Explain  F7=Specify  F8=Process  F10=Reset selections  F11=eXit
------------------------------------------------------------------------
                    Enhanced Repository Search Fields
--------+---------------------------------------------------------------
Selected|SEARCH FIELD (Current Search Value)
--------+---------------------------------------------------------------
        |AUTHOR
        |BOUNDING_INFO
        |CLASS
        |COMPONENT_STRUCTURE
        |CONCURRENCY_INFO
        |DATE_ENTERED
        |DEFAULT_VERSION
        |FUNCTION
        |FUNCTIONAL_AREA
        |ITERATION_INFO
        |LANGUAGE
        |MEDIA_MANAGEMENT
        |MEDIUM
        |NAME
        |OBJECT
   *    |OF_TYPE (Eq SOFTWARE)
```

Figure A-3: Software Search Fields User Interface

Appendix B. Supplying Components

Component Supplier Procedures

The procedure to follow for contributing a component is quite simple; unfortunately, for large components it takes a fair amount of time. The basic steps are:

1. *Collect the information as defined in data requirements.*

   The data requirements are described in the following paragraphs. This information permits the supply capability to prepare a component entry in the Repository database.

2. *Copy the component parts into your user directory on the Repository.*

   You will have to define file names in your user directory that are meaningful for you and match the component needs. These file names will be needed by the supply facility.

3. *Invoke 'Component Supply' and follow the instructions.*

   The initial form used in collecting the required information is presented in Figure B-1.

```
                    IBM TEAM STARS REPOSITORY - ASSET SUPPLY
------------------------------------------------------------------------
 Complete  Describe  AddValue                        Help    eXit
------------------------------------------------------------------------
  Attributes                Values
------------------------------------------------------------------------
name
asset_type
author
description
release_date
version
domain
function
object
organization
contact
keyword
content
language
------------------------------------------------------------------------
 Press °Return  to select attribute.
```

Figure B-1: Component Supply Initial Form

Component Data Requirements

The following is a general list of of information that should be provided when a component is submitted to the filtered repository. The specific requirements are contained in the 'Component Supply' forms.

- Component Description

- Supplier/Ownership Information
- Historical Information
- Component Relationships
- Taxonomic Attributes
- Restrictions/Limits
- Legal Information
- User/Testing Instructions

The key piece of information is ownership. If you are the author, then most likely you are the owner, or at least the organization you work for is the owner. It becomes more difficult when the component is a mixture of your work and others. Ownership may not be easily established and may take a legal opinion. It is even more difficult to establish ownership when you are supplying a component from another source (referred to as 3rd party supply). This type of supply is discouraged.

Component Coding Guidelines

Detailed guidelines for coding Ada in a reusable style are contained in the *STARS Reusability Guidelines* [IBMCode]. These guidelines were originally published in the *Consolidated Reusability Guidelines* [IBM380]. Essentially, the guidelines are the same as the guidelines which were collectively established by the STARS prime contractors in the STARS Q-Increment. Some of the original guidelines were modified for clarity and depth of definition. A few were eliminated due to experiences and comments since they were published. The guidelines have also been organized to match the metrics produced by AdaMAT.

The guideline document contains a checklist of the guidelines which is very helpful during code development and review. This same checklist is applied to the component code during component filtering by a Topic Specialist.

Coding style is distinct from code format. Code format effects readability but not reusability to any great degree. Code format can be easily handled by formatting tools. Coding style addresses the language elements you use to express the program design. It has an impact on reusability, especially with regard to maintenance and program understanding. But even coding style is not as important to reusability as compilation correctness and documentation completeness.

Col. Whitaker established the STARS program philosophy on coding style with the following note:

> STARS does not wish to impose an excessive or restrictive style on the programmer. A sensible attention to readability and portability should be sufficient guide.
>
> STARS style recommendations have to be consistent with the widest variety of operations, including the thousands of individual shops which may have local ideas, restrictions, and formats enforced by local methods. STARS, therefore, is not restrictive without compelling reason, especially in those areas where it possible to machine restructure the code to any desired style.
>
> STARS sets no specific formatting requirements, as a matter of principle. The philosophy is that one might expect to receive code from various organizations with different ways of doing things. The government will pretty-print to Ada LRM style. The only style limitation is that one should not attempt to encode information (e.g., into the case of identifiers, since Ada is case insensitive), or use other non-Ada conventions. The government should be able to restructure and extract code information that is processable by an Ada compiler.
>
> STARS is trying to develop a software technology to be used by the DoD, not just to control a small group of in-house programmers. The government should not over-specify those things it can easily adapt. Style guidelines that impose more rigid formatting rules are officious pedantry, but very common. Conventions like "_TYPE" may be used by some groups; STARS would not interfere, nor would it attempt to impose them on anyone else.
>
> Arbitrary restrictions to the full capability of Ada (such as unnecessary injunction against "use") are inappropriate. Each local shop may, for its own reasons, add additional restrictions, although STARS would recommend against anything that would limit the expressiveness of Ada. Examples of oppressive limitations include: no "function" in Ada PDL so it can be mapped to COBOL; no

"if" nested under an "if", because a tool was derived for a language without "elsif"; forbidding the use of "use", thereby denying much of Ada overloading; forbidding the "while" construct in favor of loops with exit.

STARS is experimenting with using SGML encoding for program prologue information so it can be computer processed. This documentation technique is considered separable from "Ada style", and would be the subject of other guidelines.

Appendix C. Evaluating Components

Repository Content Management

The Repository content is managed with three levels of component quality: organized, filtered, certified.

Organized Level (Entry to Repository)

Requirements

To be admitted to the organized level of the repository, a component must meet certain minimal data requirements and the information must be entered in the supply form. The component must be classified within the standard taxonomy used by the repository.

Procedures

The repository librarian supervises the admission of a component into the repository.

Filtered Level

Requirements

A component attains the filtered level of the repository through the filtering process. The component must meet more stringent data requirements and undergo analysis. The component must be evaluated for reusability and portability. They must be assessed for their conformance to the coding guidelines prescribed for reusable components.

Procedures

The librarian supervises the filtering process. Topic specialist are brought in to evaluate and analyze components. Specific attributes of the component are determined and recorded. Automated tools, such as AdaMAT, are used where appropriate.

Certified Level

Requirements

The certified level is attained when a component is determined to be 'correct'. Currently, the technology in this area is not refined and the requirements are not established.

Procedures

A component is certified through a process that includes analysis and testing. The component is placed under strict configuration management following the certification.

Component Evaluation Filters

Filtering of components in the repository is the process of reviewing components (and their parts) in order to establish a defined level of understanding about the component. For each filtering process a component attribute or report is recorded and placed in the repository.

The current filters are described in the following paragraphs in the order of their application.

1. COMPILATION

   To pass the compilation filter all the Ada parts of the component must be compiled by the VAX Ada compiler. The success of this effort is recorded in a component attribute. A list of 'withs' outside of the component and a compilation order list must have been supplied when the component was 'organized' or this filter will fail.

2. DOCUMENTATION

   To pass the documentation filter, the existing documentation parts are read and evaluated by a topic specialist. A report on the evaluation of the documentation may be written and added to the component. The general filter is a pass or fail answer to the question, "Is the document sufficient to support component reuse?"

   The documentation will be subjectively evaluated for consistency, clarity, completeness, and correctness. The documentation may be checked for spelling and grammar using automated checkers. The automated checkers also permit metric evaluations, such as 'grade level' and 'word counts'.

   If 'tagged' or pre-formatted documentation exist, it will be passed against the appropriate processors. For example, SGML tagged documentation will be passed through an SGML processor, and PostScript output will be sent to a PostScript printer.

3. METRICS

   AdaMAT will be used to establish the metrics of a coding component. A report will be added to the component that contains a roll-up of all the Ada parts. Reports may be prepared for each Ada part. While there is no specific criteria, components that display very low values may be considered for removal from the Repository.

4. REUSEABILITY

   This filter requires the application of the current STARS reusability coding guidelines [IBMCode] to the component. It is a subjective review which can be augmented with references to the AdaMAT metrics. The report is for the whole component.

   There is no specific level, rather a pass or fail is recorded for each class of reusability guidelines. The coding issues are subordinate to the overall issues of basic acceptability.

5. TESTING

   To exercise this filter, the component must be executed using the testing parts in the component. If the component has no testing parts, then it can't pass this filter. In some cases it may be reasonable to create testing parts, especially for good components where the supplier didn't contribute any testing parts. Ideally, the report is a test report with "successful" marked against most cases.

6. SECURITY

   This filter involves manual inspection for security problems. Each component part should be viewed on-line and in printed form. All parts are currently maintained in ASCII form. The Repository doesn't support reuse of executables. The review must be noted in the component attribute.

7. CORRECTNESS

The notion of correctness permits a component to be classified as certified. There are several approaches to this subject, which are being studied under STARS tasks. Currently this filter is not available.

Topic Specialist Procedures

The skill and knowledge of the topic specialist is critical to the filtering process. The specialist must judge components according to evaluation criteria in the filtering procedures. They are not mechanical, but rather subjective. The main emphasis is on reviewing the component for its potential reuse.

This process is weakly defined at this stage of the Repository iterative life cycle. As the system matures, it is expected this area will receive more attention and process feedback.

An automation of some of the filtering process is captured in the Repository Gatekeeper facility. This facility, while only an experimental prototype, does provide some useful notions of filtering. It may be used by topic specialists or submitters to coordinate a component evaluation. The facility is invoked by entering "GATEKEEP" at the VAX command prompt. Figure C-1 is a screen print from the execution of the Gatekeeper.

```
                  To Do

 Parts of WINDOW MANAGER                  Checklist for WINDOW MANAGER
Process Checklist Browse Exit Help         Pass  Fail  Save  Exit  Help
    Type             Name                     Cohesion
COMPONENT      EXTENDED CHARACTER UT          Coupling
COMPONENT      LINKED LIST                          nting interfaces
COMPONENT      N_ARY TREE P        Tools            ing dependencies
COMPONENT      STRING BOUND  Run   Exit             pendent components reusable
PACKAGE BODY   WINDOW MANAG Browse                  ts
PACKAGE SPEC   WINDOW MANAG AdaMat                  gram documentation
SEP PKG BODY   WINDOW MANAG Review AdaMat Report    anonymous types
SEP PKG BODY   WINDOW MANAG Compile                  use limited private types
SEP PKG BODY   WINDOW MANAG SGML                    constraints on numeric type
```

Figure C-1: Gatekeeper Facility

Appendix D. IBM STARS Repository System

Repository Design

The design of the Repository followed the STARS basic integration model and functional interface standards as defined in the Q-increment [IBM70]. A portability layer was defined that permits migration of the system to other Ada environments. Concessions were made to non-portable development involving the file system, the database manager SQL interface, and the database screen interface aid. The following diagram expresses the general architecture.

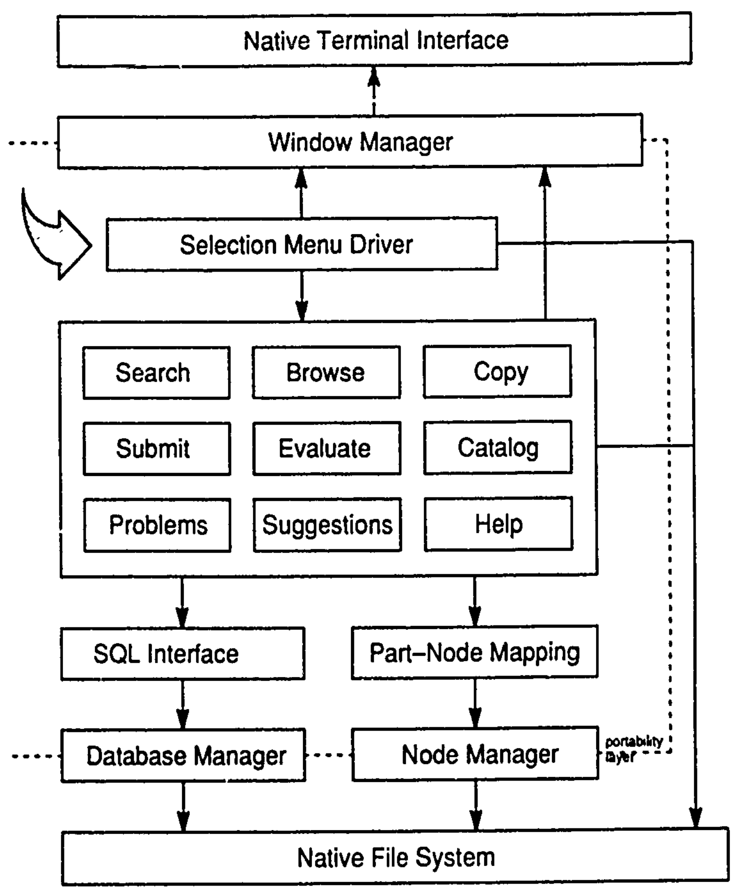


Figure E-1: Repository Basic Architecture

Repository System Facilities

The IBM STARS Repository runs on the following system:

- Hardware System
  - DEC microVAX 3600 with 32 megabytes of memory
  - 1.2 gigabytes of disk storage
  - 18 dial-in modems (16 at 2400 bps, 2 at 9600 bps)
    - ▲ 16 at 2400 bps, MNP Level 5 error correction/compression
    - ▲ 2 at 9600 bps, MNP Level 5 error correction/compression
  - high speed link to Gaithersburg, 9.2 kbps
- Software System
  - operating system: VAX/VMS Version 4.7
  - database management system: Oracle Version 5.1
  - electronic mail: VMS Mail Version 4.0
  - electronic conferencing: ANU NEWS Version 5.8
  - file transfer: Kermit-32 Version 3.2.077
  - Ada compiler: DEC Ada for VMS Version 1.4
  - Ada analysis: DRC AdaMAT Release 2.0

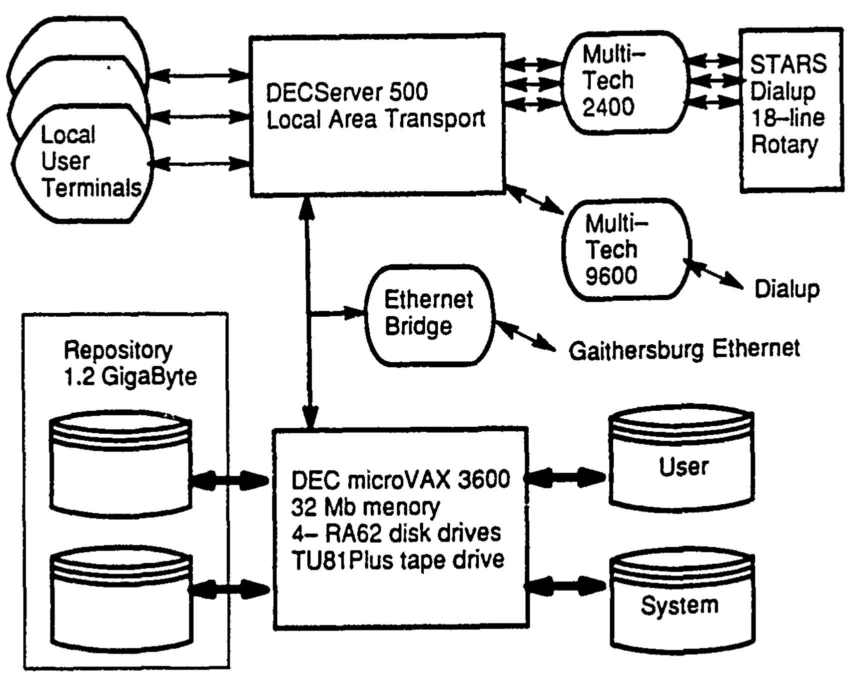


Figure E-2: Repository System Configuration

Repository Capabilities

The following is a list of the capabilities of the IBM STARS Repository [IBM110]:

- Support for Reuse Library
  - component search/select
    - ▲ by hierarchy class
    - ▲ by facet
    - ▲ by attribute

- component browse (examine)
- component copy (extract)
- component tracking
- component subscription
- component problem reporting
- component submit
- component catalog generation
- component catalog browse

- Support for Software Engineering
  - Ada compilation
  - Ada metric analysis
  - SGML document preparation

- Support for General Information
  - directory search
  - file browse
  - electronic mail
  - electronic conferencing
  - file transfer (upload/download)
  - people search (scan)
  - system problem reporting
  - usage statistics browse

Repository Database

Information Model

The heart of the Repository is the database and the information model that it implements. The IBM STARS Repository information model is based on a 'component' and its 'parts'. The model is currently being revised, but the following diagram of a recent model presents the basic concepts.

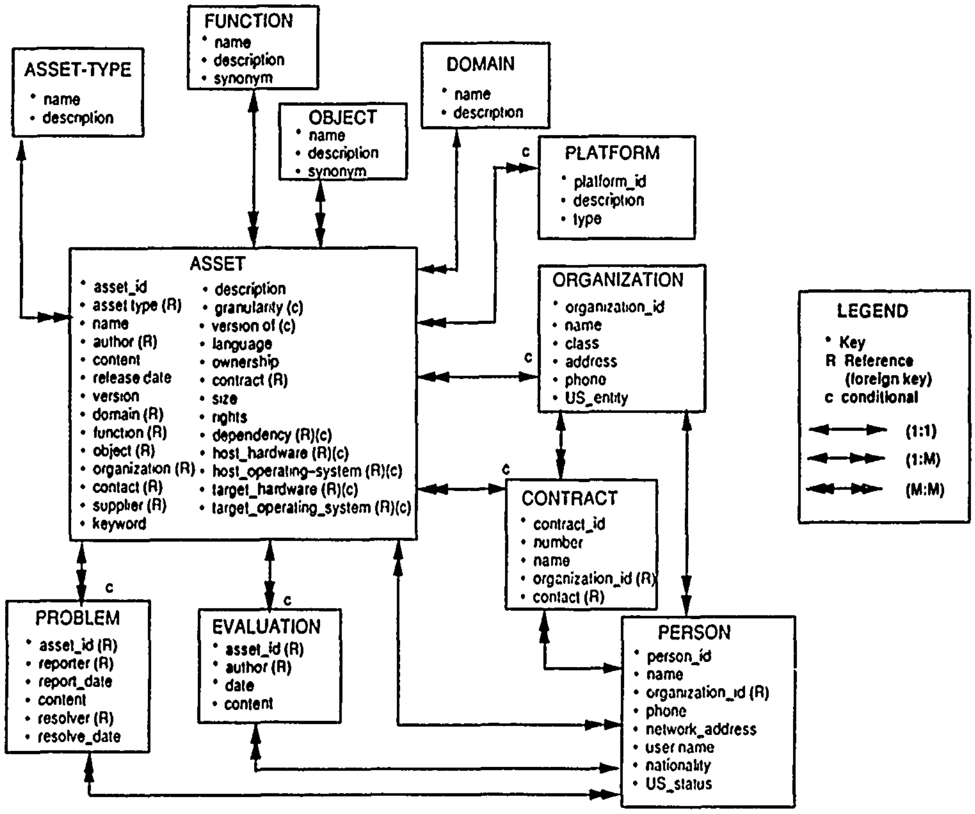


Figure E-3: Repository Information Model

Fig 7

Database Table Definitions

In using and administrating the repository, you will frequently need to know the permitted values (or vocabulary) for particular component attributes. They are presented in this document for users to develop an understanding of the type and depth of the table content. These values are stored in the Repository database and are modified over time as appropriate. All processes in the Repository system that use these values, retrieve them from the current database. To view accurate actual values, you must refer to the current database.

Part Types

The following is a list of PART TYPES available in the Repository.

DESCRIPTION	Description of the component
HISTORY	History of component including author, company, address, etc
DESIGN	Design notes for component
REQUIREMENTS	List of modules or building blocks necessary to run
UNITS	Package name of component
DOSFILE	Name of DOS file in which the component's code is stored
RIGHTS	Copyrights and release agreements
SPEC	Ada specification for the component
BODY	Ada body of code for the component
TEXT	File containing textual characters, intended to be viewed
SGML	File intended to be viewed via some SGML processor

Facets

The following is a list of FACETs and FACET TERMs in the Repository [IBM1580].

```
FACET            FACET TERM        ALIASES
---------------  ---------------   ---------------
FUNCTION         ANALYZE           EVALUATE
                 CHECK             VERIFY
                 DECODE            TRANSLATE
                 EXAMINE           DISPLAY,BROWSE
                 FORMAT            TRANSFORM
                 MANAGE
                 MANIPULATE        APPEND,EXTRACT
                 MODIFY
                 USE

OBJECT           DATES             CALENDAR
                 EXTENDED_CHARS
                 FILES             SOURCE,TEXT
                 LISTS
                 STACKS
                 STRINGS
                 TREES             GRAPHS
                 WINDOWS

MEDIUM           ARRAYS
                 FILES
                 RECORDS

SYSTEM_TYPE      FILE_HANDLER
                 SCHEDULER
                 USER_INTERFACE

FUNCTIONAL_AREA  DOS
                 SWE

SETTING          SWE
```

As the repository grows and the contents become more diverse, the Facets, Terms, and Aliases will be re-examined to be consistent in the meaning of terms and the scattering effect of limiting the number of aliases used. If the composite terms begin to make too large a selection-set then remove one or more aliases and select one for a new primary term. Components being added can then be tied to the new, primary term and all existing components that better fit the new term can be linked to the new term.

Component Classes

The following is the high level list of the hierarchical component CLASS values in the Repository. Each class several several levels of subclasses. This is the basic taxonomic structure of the Repository content.

```
0   STARS PRIMES - INTERCHANGE
1   STARS Enhanced Repository Taxonomy
2   BOEING DELIVERABLES
3   IBM DELIVERABLES
4   UNISYS DELIVERABLES
5   STARS FOUNDATIONS (by Contractor)
6   NOSC_WIS (by Contractor)
7   SIMTEL20
8   CAMP (Common Ada Missle Packages)
9   SDME (S/W Dev. & Maint. Env.)
10  Etc.
```

Development Guidance

The Repository design and implementation followed several general sets of guidance. The development followed the IBM STARS Quality Assurance and Configuration Management Plan [IBM1320]. To review the code work products refer to the in the code delivery of the STARS IR40 task [IBM1600].

Typical User

The Repository was designed with the following typical user in mind.

- Little knowledge of the Repository host computer or operating system.
- No knowledge of the Repository database and approaches to database access.
- Reasonable knowledge of Ada.

User Interface

The basic reference for the Repository user interface is the IBM System Application Architecture (SAA) - Common User Access (CUA) [CUA89]. It defines the concepts of windows, title bars, action bars, client areas, buttons, emphasis, and keyboard interactions. The CUA is generally applied to pixal-graphic user interfaces, but we have generalized the to guidance to character graphics because of the limited capabilities of the Repository system and the remote user systems. The user interface also incorporates models and ideas found in character graphic tools commonly used on VMS.

The following are some guidelines used to develop the user selection menus:

- Selection items should be based on the repository capability list.
- All menus should have "Report Problem", "Help", and "Exit".
- Selection items should only exist in one place of the selection menu hierarchy (except for the above items).
- Selection items not implemented should not be on the menu.
- Selection items should be a 'verb' acting on a 'noun' (examples: 'search directory' and 'browse file'.
- Selection items should be upper and lower case, with major words capitalized, and minor words in all lower case.
- The selection character should be the initial character of the 'verb'.
- The selection character should be emphasized and in upper case.
- Submenus should be titled with the higher level menu selection item.
- Help information should be context sensitive.
- Function keys should not be used.
- Common sense and user perception should prevail when these guidelines produce 'funny' situations.

Object Names

The naming of objects (files, tables, commands) has a significant effect on the management, readability, and reusability of a system. For the Repository development, we have established the following naming guidelines:

- Eliminate all file or directory names in code. Try to acquire the names from standard input.

- When it is not possible to eliminate imbedded file names, isolate the names in easy to locate and modify places.

- Do not use dates, version numbers, CDRL numbers, or other project specific names. Name items according to what the item does, and not according to where it is. Names like DEMO2, STARS_PRIME, or REPOS89 inhibit reuse. Names like REPOS_SEARCH and WINDOW_MANAGER are much better.

- Don't use plurals for names (ie use COMPONENT table, not COMPONENTS table; use COLOR type, not COLORS type).

Index

Abstract

This Guidebook provides high-level information for all users of the IBM STARS Repository. It describes how to use the Repository system in the context of software reuse.

The Guidebook is organized according to specific roles that users perform when using the system. It addresses the information needs of component reusers, component suppliers, and repository administrators.

This Guidebook defines

- How to use the IBM STARS Repository,
- Resources and capabilities of the Repository,
- Component standards for admission to and promotion within the Repository,
- Procedures for submission and usage of repository components,
- Processes involved in managing repository assets, and
- The reuse process in the context of the STARS repository.

The Guidebook is the key document in a suite of Repository guides. The other documents are the *STARS Reuseability Guidelines*[IBMCode] and the *IBM STARS Repository User's Guide*[IBMUser]. These documents were separated from the Guidebook for practical reasons, but they are required to make full use of the Guidebook.

Preface

The content of the Guidebook is tightly coupled to the IBM STARS Repository. It will be reviewed periodically, incorporating the lessons learned from using the guidelines and procedures, and updating as the Repository matures. In the short term, the Guidebook will help meet the practical needs of the IBM STARS development teams. In the long term, it will serve as a baseline element in a more automated software development environment.

The guidelines and procedures in the Guidebook apply to all IBM STARS team tasks. In particular, they apply to Ada code and related documentation as it is admitted to and managed within the Repository. The Guidebook serves as a starting point for development in future increments of the STARS project.

The term 'Repository' has developed several meanings within the software development industry. In the STARS project, the term has been associated with a machine, a facility, and an application. In general, this Guidebook uses the latter meaning, also referring to the Repository as a *Reuse Library*.

This Guidebook was developed by the IBM Systems Integration Division, located at 800 North Frederick, Gaithersburg, MD 20879. Questions or comments should be directed to the author, Robert W. Ekman at (301) 240-6431, or to the IBM STARS Program Office.

IBM STARS Repository

Guidebook

July 2, 1990

Contract No. F19628-88-D-0032
Task IR40: Repository Integration

Delivered as part of:
CDRL Sequence No. 1550

Prepared for:

Electronic Systems Division
Air Force Systems Command, USAF
Hanscomb AFB, MA 01731-5000

Prepared by:

IBM Systems Integration Division
800 North Frederick
Gaithersburg, MD 20879

Contents

1. Introduction

Welcome to the IBM STARS Repository.

The IBM STARS Repository facilitates the distribution and sharing of software, documentation, and project information within the STARS community [RFP87]. It also serves as a bridge between the STARS community and the software engineering industry.

In particular, the Repository

- stores and organizes designated material,
- provides capabilities for convenient searching and retrieval,
- provides capabilities for electronic dialogue (e.g., mail, conferencing, etc.) among its user community, and
- provides 24-hour access for convenient delivery or copying of material over government or public electronic networks.

In addition, the Repository

- is an instance of the STARS software engineering environment, and
- presents a vision of the software reuse process model.

The Repository supports

- software engineering through the software reuse model,
- project coordination through work product sharing and prototyping, and
- contract administration by archiving deliveries.

This Guidebook is your introduction to the IBM STARS Repository. It gathers in one place all the references, guidelines, and procedures related to the operation and use of the Repository [IBM1540]. It will give you an understanding of the nature and scope of the Repository.

After skimming this guidebook, you should use the Repository system. Only through hands-on exercise of the system will you become comfortable with it. A section in the Appendix of this document describes how you can get an account and access the system.

The Repository is a maturing system with a component supply and problem correction process. You will notice improvements in the Repository over what is described in this Guidebook. The component content of the reuse library is dynamic. New components are added every week and existing components are being reviewed.

You are encouraged to make the most out of the system and its features, and to report problems and suggestions. The Repository support team is available and willing to help.

Reuse and the Repository

A repository is the primary support tool for software reuse and is a key element in project life cycle integration SEI89. Without some form of a repository or reuse library, significant reuse on a software development project is impossible. Reuse starts during early phases of the project [RSL87]. You must set goals for reuse and commit to not "reinvent the wheel."

Your project's problem definition and solution is related to a particular domain or set of domains, such as 'software engineering' or 'air traffic control'. Likewise, repositories are related to specific domains. To be successful with reuse, you must locate repositories that support the domains related to your project. You must become familiar with the repository access and content as early as possible.

The system engineers must define architectures with a view toward reuse. Requirements and specifications must be defined recognizing the availability and capability of reusable components. Reuse must be specified. If you start building without reuse specifications, you cannot take advantage of the significant quality and cost benefits of software reuse.

During implementation you bring together the components from the repository. Large grain components are most desirable. They will form the major parts of the solution. Small grain components are incorporated while editing and developing new system elements. During new code creation, you should keep open access to the repository. This will allow you to interrogate the repository for candidate components as you develop code.

As you complete your project or incremental release, you should consider putting key components from your development into the repository. If you made modifications to repository components, then you should return them as new versions. The reuse-repository model is a cyclic process, with supply and consumption the driving forces.

The basic model of the Repository system consists of three user environments: the repository platform, the software engineering environment, and the external casual user system. The repository platform consists of two major elements: the Repository application and the user communication mechanisms. The software engineering environment consists of a collection of tools coordinated through an object manager. The external user system provides access to the repository elements from outside the software engineering team. These elements and their relationships are presented in Figure 1.

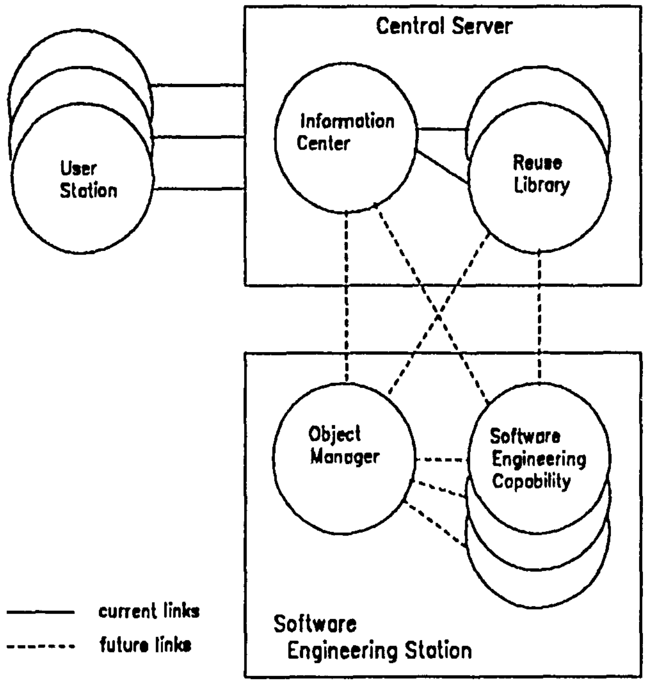


Figure 1-1. Basic Repository System Model

Definitions of Key Terms

In order to understand the processes described in the Guidebook and represented in the Repository implementation, you should review the following terms and definitions. They will help you relate this system to your previous experiences in libraries and software development. They are in an order that permits sequential reading, rather than an alphabetic order. Additional terms used in the Guidebook and the Repository system are defined in 4, "Glossary" on page 4-1.

component — A collection of related work products to be used as a consistent set of information. Software work products can include specifications, design, source code, machine code, reports, compilation units, code fragments and other components. It is the primary object type that is the concern of a repository. Also referred to as an **asset**, or **resource**.

part — An element of a component, such as design documents, source code, test information, and data rights. While a part may be copied or browsed, when stored in a repository it is always associated with a component.

repository — An element of the software engineering environment in which software work products and information about them are stored. Its primary purpose is support of reuse. Also referred to as a **reuse library**.

repository system — An instance of the software engineering environment, including computer hardware, operating systems, software tools, standards, and procedures that together provide a complete repository capability. These capabilities include acquiring, storing, managing, retrieving, and dispensing software work products and information about them. A repository system may contain several repositories, each dealing with a different problem domain.

depository — A repository or part of a repository whose contents are deposited as is, with few or no constraints. A depository is primarily a storage facility. Quality and usability are unpredictable; the burden is on the user to find and evaluate useful items.

organized repository — A repository whose contents are documented and organized in a comprehensive manner as components and parts. Finding, reviewing, and extracting components are supported. The quality and usability remain unpredictable.

filtered repository — A repository whose components are subjected to standards on form, content, quality, and consistency. This is not a physical partition from the organized repository, rather it is a documented higher level of confidence in an existing organized component.

certified repository — A repository whose components exhibit the highest level of confidence. A certified component permits the proving of correctness in a solution that reuses the component. Aspects of security, ownership, distribution, and user set take on greater importance.

reuser — A person who reviews the contents of a repository and extracts components for reuse. The common user of a repository. Also referred to as a **user**.

supplier — A person places components into a repository. Also referred to as a **contributor**, or **submitter**.

librarian — A person who coordinates, organizes, and controls the contents of a repository. Also referred to as a **system administrator, database administrator**, or **repository administrator**. This person frequently provides a first level help for other users.

topic specialist — A person who is responsible for the evaluation of components and promoting them to the filtered of certified status. This person understands the repository domain and has reviewed many of the components in the repository. Also referred to as a **domain expert**, or **technical consultant**. This person frequently provides a second level help for other users, via reference from the librarian.

component data file A file containing structured information about the component. The file is prepared automatically, based on answers to questions in the on-line supply process. It is the representation of the component when it is held in the librarian's 'holding bin'. It is reviewed by the librarian and facilitates the entry of the component into the repository. This file also forms the basis for component interchange with other repositories.

filtering The act of evaluating components and promoting them to filtered status.

certifying The act of certifying components and promoting them to certified status.

gatekeeper An automated capability that facilitates component evaluation. It is usually associated with entry to the repository and filtering, but may be used by suppliers to review their components before submitting to the repository, or as support for the certification process.

2. Repository User Roles

In using the Repository facilities and services, you and other individuals acting in various roles will interact to accomplish reuse. This guidebook is organized on the basis of these roles. For each role, references are made to the guidelines and procedures to be used in conducting your role. Each role carries with it duties and responsibilities that combine to make the repository mission a success.

It is important to realize that the role is distinct from the individual or individuals who perform in that role. It is possible, for instance, that a given role may be performed in whole or in part by a single individual, by more than one individual, or by several individuals working together. Individuals often perform several different roles when using the Repository.

Component Reuser

A *reuser* is a person who reviews the contents of a repository and extracts components for reuse within their development project. The satisfaction of the reuser is the primary focal point of the IBM STARS Repository. Through a large component supplies and effective retrieval mechanism the Repository will increase the user's degree of reuse.

Reuse is a simple, easy to accomplish process, but requires a confidence and commitment on the part of the reuser. First, you establish a general requirement for some form of reusable component. You express this need in general textual terms, such as 'window system', 'list processor', 'memory manager', or 'file browser'.

The next step is to access the Repository and invoke the component search facility. Using the menu interface, you construct a query that is broad enough to encompass many candidate components, but restrictive enough to eliminate obvious non-candidates. You then invoke the query, review the returned candidate list, and select specific components for extraction. You may want to browse the components from the candidate list to review the component before extraction.

In the extraction process, you will create copies of the selected components in your local directory on the Repository system. You then copy them to your development system. Once in your development system, you may compile the components and test them.

You may locate several components that meet your needs. You should consider extracting all of them and attempt to build a set of alternative solutions. In doing assessment of the alternatives you will develop a better understanding of your problem domain and will gain material for comparative analysis.

In using the Repository, the reuser takes on some responsibility. The reuser must follow the code of conduct as defined by the Repository administration [IBM1460, IBM1470]. Reusers should not redistribute components outside their project. If other people or projects want some of the Repository content, they must establish their own contact into the Repository. When reusers discover errors with a component or has problems with the Repository system, they should report them through the on-line problem reporting mechanisms.

Component Supplier

A *supplier* is a person who contributes components to the Repository. All reusers are potential suppliers, but a supplier is a reuser with a component that could be reused by another developer. The supplier provides feedback into the reuse process by first extracting and subscribing to components, and then by improving the component or developing new components and submitting them to the Repository

The supplier should be the author of the component, because the author will know the component attributes and be able to fill out the supply forms accurately. The supplier will become the point of contact for the component. Third party suppliers are not encouraged.

While it sounds simple, the supply process is complicated by data rights, ownership, incentives to supply, and amount of effort to supply. The effort to supply is inversely proportional to the reuse effort, and according to the economics of the process, it should be more difficult to supply than reuse. The IBM STARS Repository has not resolved these issues, but has provided a rudimentary facility in which to prototype the operational aspects of the issues.

The role of the supplier involves packaging and presenting work products for inclusion in the STARS Repository. The work products should adhere to the STARS guidelines for reusability and portability. You should pay particular attention to the issues of adaptation and tailoring of your components. If you are developing software and want more information on the STARS coding standards, refer to the *STARS Reusability Guidelines*[IBMCode*rbrk.. You may also want to run your code through the an Ada analysis tool, such as AdaMAT which is available on the IBM STARS machine. The issues of coding standards should not deter you from submitting a component. The evaluation process of filtering will review the component for style and standards compliance. If you believe your component is reusable, then you should submit it.

As a supplier, you are responsible for ensuring that the component information requirements are satisfied prior to submitting work products to the Repository. This usually involves collection of detailed information about the component to be supplied. You will have to transfer the component parts to the Repository system. You then invoke the component supply facility and follow the on-line directions.

You may want to submit a small test component using the on-line supply facility first. You should take screen prints of the supply questions so that you can compile the required information off-line before you start with a 'real' component. The Repository Librarian will recognize your 'test' component and not place it in the Repository.

Developers in the STARS project form a special class of suppliers. All STARS developers should plan on becoming suppliers sometime during their development cycle. In adhering to the STARS guidelines and contract requirements, the STARS developers will produce products that are inherently ready for supply.

Repository Administration

Repcsitory Manager

The *repository manager* owns and operates the Repository. The manager is responsible for establishing and maintaining Repository procedures and policies. The manager has overall responsibility for the Repository Librarian and Topic Specialists.

The manager has the following specific responsibilities:

- Maintain system availability
- Manage and review access control
- Store STARS contract delivery items
- Conduct system backup
- Implement disaster recovery and vital records plans

Repository Librarian

The *repository librarian* coordinates, organizes, and controls the contents of the Repository. The librarian acts as a database administrator. The librarian is available for help with reuse and supply problems.

The librarian has two different paths to follow, depending on the type of item submitted to the system. Figure 2 is a graphic illustration of these paths.

- Review submitted reusable material in the 'holding bin' and place the material in the organized repository, or
- Copy STARS contract deliverables to the depository with minimum inspection.

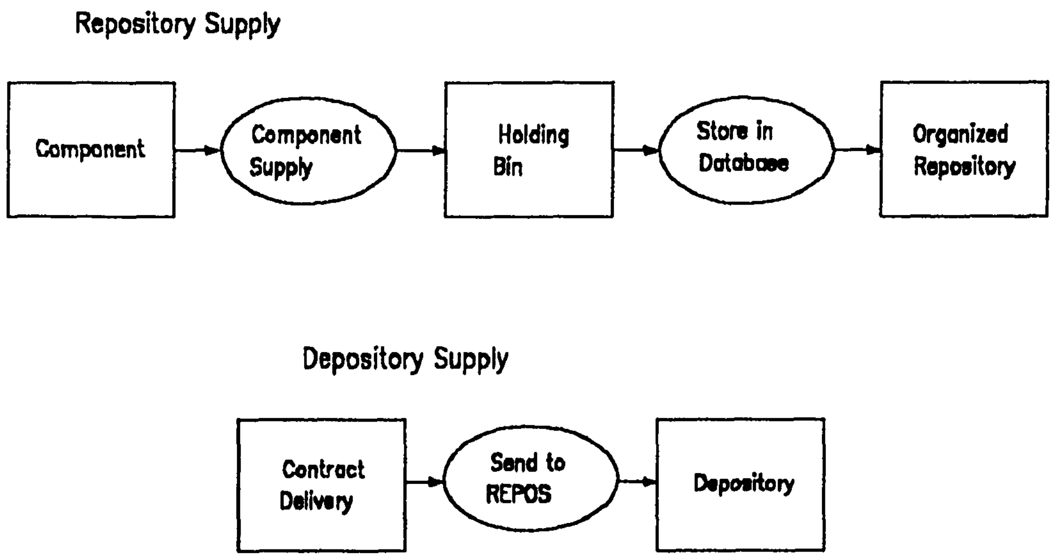


Figure 2-1. Repository Librarian Activities

When a component is placed in the 'holding bin', it is represented by a component data file. This file is reviewed by the librarian and may modify the file to bring the component into compliance with entry criteria. This form of the component is also used when exchanging components with other repositories.

Topic Specialist

The *topic specialists* review and evaluate components. They understand the repository domain and have reviewed many of the components in the repository. They are responsible for the evaluation of components and promoting them to filtered or certified status. The topic specialist frequently provides a second level help for other users, via reference from the librarian.

The act of evaluating and promoting components is referred to as 'filtering'. Specific filtering guidance is presented in the Appendix. The specialist uses an automated evaluation facility called the 'gatekeeper'. This facility is also available, as a prototype, for component developers and the librarian to evaluate components within their part of the reuse process.

The specialist role supports quality assurance of work products that are received into the repository by ensuring that the Repository reusability guidelines and portability guidelines are met. In addition, the specialist contributes to Repository configuration management by assuring that the work product is properly classified and properly inserted into the Repository structure.

3. References

There are many exciting successful reuse efforts completed or in-progress. All the major DoD software agencies and contractors are involved. Of particular interest are the McDonnell Douglas CAMP experiences [CAMP86] and the SEI Software Reuse Project [SEI89]. A literature search in your project's problem domain will turn up references appropriate for your project.

The following is a list of references in this Guidebook and its appendices in the order that they are cited. References that are exclusively used in the reusability coding guidelines are contained in that section only. The IBM STARS documents with reference tags of [IBMxxx] are available in electronic form from the depository.

[IBMCode] IBM Systems Integration Division, *STARS Reusability Guidelines*, April 30, 1990.

[IBMUser] IBM Systems Integration Division, *IBM STARS Repository User's Guide*, April 30, 1990.

[RFP87] United States Department of Defense, Department of the Air Force, *STARS Competing Primes Lead Contracts Request For Proposal*, F19628-88-R-0011, November 5, 1987.

[IBM1540] IBM Systems Integration Division, *Repository Guidebook (Draft)*, CDRL Sequence No. 1540, September 14, 1989.

[SEI89] Software Engineering Institute, "Reuse: Where to Begin and Why," *Affiliates Symposium*, May 2-4, 1989.

[RSL87] Burton, B. A., and others, "The Reusable Software Library," *IEEE Software*, July 1987.

[IBM1460] IBM Systems Integration Division, *Draft Policies and Procedures*, CDRL Sequence No. 1460, January 19, 1990.

[IBM1470] IBM Systems Integration Division, *Repository Operations and Procedures*, CDRL Sequence No. 1470, March 7, 1990.

[CAMP89] McDonnell Douglas Astronautics Company, "Overview and Commonality Study Results,," *Common Ada Missile Packages (CAMP)*, AFATL-TR-85-93, May 1986.

[Peterson79] Peterson, A. S., "Coming to Terms with Terminology for Software Reuse," *Reuse in Practice Workshop*, 1989.

[IEEE729] IEEE, *Standard Glossary of Software Engineering Terminology*, ANSI/IEEE STD 729-1983, IEEE Standards Board, September 23, 1983.

[Webster88] Merriam-Webster Inc., *Webster's Ninth New Collegiate Dictionary*, Springfield Mass., 1988.

[IBM1440] IBM Systems Integration Division, *Practical Aspects of Repository Operations*, CDRL Sequence No. 1440, January 10, 1990.

[IBM380] IBM Systems Integration Division, *Consolidated Reusability Guidelines*, CDRL Sequence No. 0380, March 21, 1989.

[IBM70] IBM Systems Integration Division, *Consolidated Technical Development Plan for STARS Competing Prime Contractors*, CDRL Sequence No. 0070, November 11, 1989.

[IBM110] IBM Systems Integration Division, *Environment Capability Matrix*, CDRL Sequence No. 0110, March 17, 1989.

[IBM1580] IBM Systems Integration Division, *Taxonomy Report*, CDRL Sequence No. 1580, January 19, 1990.

[IBM1320] IBM Systems Integration Division, *Quality Assurance/Configuration Management Plan*, CDRL Sequence No. 1320, October 20, 1989.

[IBM1600] IBM Systems Integration Division, *Version Description Document for the IBM STARS Repository*, CDRL Sequence No. 1600, January 31, 1990.

[CUA89] IBM, *Common User Access Advanced Interface Design Guide*, SC23-4582-0, June 1989.

4. Glossary

The terms and definitions used in the IBM STARS Repository are listed in "Definitions of Key Terms" on page 1-3. The following terms and definitions provide additional help in describing software reuse. They are extracted from the "Partial Glossary for Software Reuse," contained in a paper by A. Spencer Peterson (SEI) titled "Coming to Terms with Terminology for Software Reuse" [Peterson89].

The terms and definitions are taken from a draft update to ANSI/IEEE Std 729 (Glossary of SW Terminology) [IEEE729], except where the term is marked with an (M) for Modified where inserted text is enclosed in [ ], or with a (*) signifying a term that is not defined in the IEEE draft. Other comments are enclosed by { } and placed at the end of the definition.

abstract data type: A data type for which only the properties of the data and the operations to be performed on the data are specified, without concern for how the data will be represented or how the operations will be implemented.

abstraction: (1) A view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information. (2) The process of formulating a view as in (1).

adaptation data(M): Data used to adapt a program [or component] to a given installation site or to given conditions in its operational environment.

adaptation parameter(M): A variable fflor placeholder" that is given a value fflor other appropriate information" to adapt a program fflor component" to a given installation site or to given conditions in its operational environment.

adaptive engineering(*) The process of modifying a system or component to perform its functions in a different manner or on different data than was originally intended.

adaptive maintenance(M): Software maintenance performed to make a computer program fflor component" usable in a changed environment.

application-oriented language: A computer language with facilities or notations applicable primarily to a single application area.

architecture: The organizational structure of a system or component.

artifact(*): Any product of the software development process.

asset(*): a set of reusable resources that are related by virtue of being the inputs to various stages of the software life-cycle, including requirements, design, code, test cases, documentation, etc. {An asset can be design and the control code for using other assets in the library in a more powerful way. Assets are the fundamental element in a reusable software library.}

component: One of the parts that make up a system. {A component is some useful portion of a computer program. It may be subdivided into other components.}

control abstraction(*): (1) The process of extracting the essential characteristics of control by defining abstract mechanisms and their associated characteristics while disregarding low-level details and the entities to be controlled. (2) The result of the process in (1).

data abstraction (1) The process of extracting the essential characteristics of data by defining data types and their associated functional characteristics and disregarding representational details. (2) The result of the process in (1).

domain(*) The set of current and future systems/subsystems marked by a set of common capabilities and data.

domain analysis(*): (1) The process of identifying, collecting, organizing, analyzing, and representing a domain model and software architecture from the study of existing systems, underlying theory, emerging technology, and development histories within the domain of interest. (2) The result of the process in (1).

domain engineering(*) The construction of components, methods, and tools and their supporting documentation to solve the problems of system/subsystem development by the application of the knowledge in the domain model and software architecture.

domain model(*): A definition of the functions, objects, data, and relationships in a domain.

functional abstraction(*): (1) The process of extracting the essential characteristics of desired functionality by defining its abstractly along with its associated behavioral characteristics and disregarding low-level details. (2) The result of the process in (1).

independence(*): The ability of a component to be used with different compilers, operating systems, machines and applications than those for which it was originally developed. Independence is closely related to *portability*.

maintainability(*): The ease of modifying a component, whether it be to meet particular needs or to fix bugs.

master library: A software library containing master copies of software and documentation from which working copies can be made for distribution and use. {This should be meticulously maintained and controlled by a special group of reuse engineers and librarians.}

modularity(M): The degree to which a system, computer program fflor code component" is composed of discrete components such that a change to one component has minimal impact on other components.

perfective maintenance(M): Software maintenance performed to improve the performance, maintainability, or other attributes of a computer program [or component].

platform(*): Platform refers to the architecture for the system for which the product is intended (hardware, operating system, and Ada compiler). Some products may be intended for several different platforms. Platforms listed should also indicate whether they are host platforms, target platforms, or both.

portability(*): The ability of an application or component to be used again in a different target environment than the one it was originally built for. The phrase *target environment* may be defined broadly to include operating systems, machines, and applications. To be ported effectively, components may need to be tailored to the requirements of the new target environment. See also *reusability* and *independence*.

production library: A software library containing software approved for current operational use.

reliability(*): The extent to which a component performs as specified. A reusable component performs consistently with repeated use and across environments (that is, operating systems and hardware).

resource(*) Any software entity placed into a software library for purposes of reuse.

retirement(M): (1) Permanent removal of a system, component [or resource] from its operational environment [or the master library.] (2) Removal of support from a operational system, component, [or resource].

reusability(M): The degree to which [a] software [resource] can be used in more than one computer program [or system, or in building other components or parts.] {See also *portability*.}

reusable software(*): Software designed and implemented for the specific purpose of being reused.

reuse(*): The application of existing solutions to the problems of systems development.

reuse engineering(*): (1) The application of a disciplined, systematic, quantifiable approach to the development, operation and maintenance of software where reuse is a primary consideration in the approach. (2) The study of approaches as in (1). {The same definition as for 'software engineering' given in the IEEE standard except for the addition of the phrase beginning with 'where'.}

software(M): Computer programs, [code components and other artifacts], procedures, and possibly associated documentation and data pertaining to the operation of a computer system [or its components].

software architecture(*): The packaging of functions and objects, their interfaces, and control to implement applications in a domain.

software library(M): A controlled collection of software [resources] and related documentation designed to aid in software development, use, [reuse], or maintenance.

software repository: A software library providing permanent, archival storage for software and related documentation. {The key word is 'archival". Also note the word 'control' is not mentioned.}

software reuse(*): (1) The process of implementing new software systems and components from pre-existing software. (2) The results of the process in (1).

specification(M): A document [or other media] that specifies, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or component, and, often, the procedures for determining whether or not these provisions have been satisfied.

tailorability(*): The ease of modifying a component to meet particular needs. It should be distinguished from *maintainability*, which includes tailorability, but also includes the idea of corrective maintenance (fixing bugs).

taxon(*): A group of resources constituting one of the categories in a taxonometric classification for reusable software in one or more domains. {The plural is taxa.} {A taxonomic group or entity [Webster88].}

taxonomy: The study of the general principles of scientific classification. [Webster88&rbrl..

5. Acronyms

The following is a list of acronym, abbreviation, and similar terms used in this Guidebook and its appendices.

Acronym	Meaning
AdaMAT	an Ada Metric Analysis Tool by Dynamics Research Corp.
ADT	abstract data type
ANSI	American National Standards Institute
CDRL	Contract Data Requirements List
CUA	Common User Access
DEC	Digital Equipment Corporation
DoD	(United States) Department of Defense
DOS	Disk Operating System (for a personal computer)
DRC	Dynamics Research Corporation
IBM	International Business Machines
IEEE	Institute of Electrical and Electronic Engineers
IR40	IBM STARS R-increment task for Repository Integration
Oracle	a commercial relational database product
RFP	Request for Proposal
SAA	System Application Architecture
SAIC	Science Applications International Corporation
SEI	Software Engineering Institute
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
STARS	Software Technology for Adaptable, Reliable Systems
VAX	a computer system from Digital Equipment Corporation
VMS	a proprietary operating system for a VAX

Appendix A. Reusing Components

Repository Access

The IBM STARS Repository is accessed remotely from your system via dial-up or Internet connections. Complete user information is found in the *IBM STARS Repository User's Guide* [IBMUser]. This guide contains sections on

- Getting an account,
- Remote user system requirements,
- Making the connection, and
- System commands.

The policies and procedures governing your use of the system are documented in the Repository Policies and Procedures [IBM1460] and the Repository Operations and Procedures [IBM1470] documents. Additional information about the Repository System can be found in *Repository Operations* [IBM1440].

Repository Menu Application

The Repository application uses a menuing system that is based on a rudimentary window manager. To display the primary menu, enter "repos" at the system command prompt.

```
IBM STARS Team Repository

Component Search
Directory Search
Component Supply
Browse Current Catelog
Repository Tools
Repository Services
Suggestion Box
Problem Reporting
Help
Exit
```

Figure A-1. Primary Selection Menu

Each menu is composed of several options including an "Exit" option. Each option may be selected by moving the highlight bar to the option of interest (using the up and down arrow keys) and pressing Enter. A second means of selecting an option is to simply enter the highlighted letter in the option of interest.

Several of the options offer submenus. Return to the next higher level of the menu system is always accomplished by pressing "x" or moving the highlight bar to "Exit" and pressing Enter.

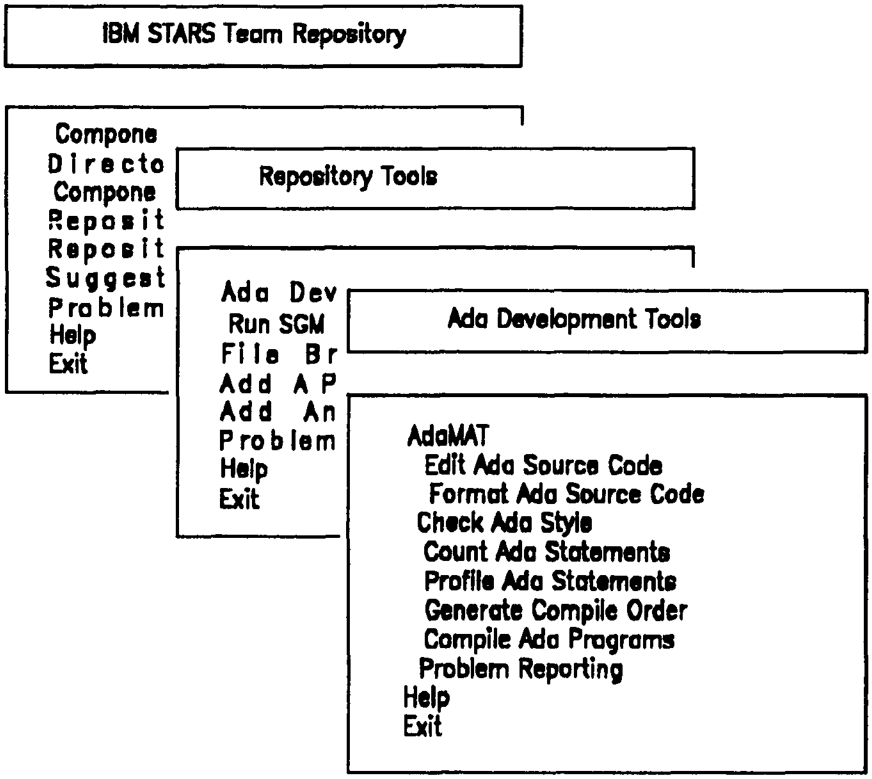


Figure A-2. Example of Submenus

The Repository application has context sensitive help information. By pressing "h" a full screen of information will be displayed describing the capabilities of the current menu.

Component Reuser Procedures

The primary Repository capability used by the reuser will be the component search facilities. At first, these facilities will seem complicated, but after a few passes through the process you will appreciate the interface and be able to locate components of interest.

There are two basic search facilities:

- Component Search - direct retrieval based on component attributes. Within the database there are two primary types of components: software and document.
- Directory Search - selection of depository directory and then files, ordered by project and tasks.

The following figure presents the user interface search fields for the software attributes in the database search facility.

F6=Explain F7=Specify F8=Process F10=Reset selections F11=eXit	
Enhanced Repository Search Fields	
Selected	SEARCH FIELD (Current Search Value)
	AUTHOR
	BOUNDING_INFO
	CLASS
	COMPONENT_STRUCTURE
	CONCURRENCY_INFO
	DATE_ENTERED
	DEFAULT_VERSION
	FUNCTION
	FUNCTIONAL_AREA
	ITERATION_INFO
	LANGUAGE
	MEDIA_MANAGEMENT
	MEDIUM
	NAME
	OBJECT
*	OF_TYPE (Eq SOFTWARE)

Figure A-3. Software Search Fields User Interface

Appendix B. Supplying Components

Component Supplier Procedures

The procedure to follow for contributing a component is quite simple; unfortunately, for large components it takes a fair amount of time. The basic steps are:

1. *Collect the information as defined in data requirements.*

   The data requirements are described in the following paragraphs. This information permits the supply capability to prepare a component entry in the Repository database.

2. *Copy the component parts into your user directory on the Repository.*

   You will have to define file names in your user directory that are meaningful for you and match the component needs. These file names will be needed by the supply facility.

3. *Invoke &csq,Component Supply' and follow the instructions.*

   The initial form used in collecting the required information is presented in Figure B-1.

```
IBM TEAM STARS REPOSITORY - ASSET SUPPLY
---------------------------------------------------------
Complete   Describe   AddValue      Help    eXit
---------------------------------------------------------
Attributes                 Values
---------------------------------------------------------
name
asset_type
author
description
release_date
version
domain
function
object
organization
contact
keyword
content
language
---------------------------------------------------------
Press  °Return   to select attribute.
```

Figure B-1. Component Supply Initial Form

Component Data Requirements

The following is a general list of of information that should be provided when a component is submitted to the filtered repository. The specific requirements are contained in the 'Component Supply' forms.

- Component Description
- Supplier/Ownership Information
- Historical Information

- Component Relationships
- Taxonomic Attributes
- Restrictions/Limits
- Legal Information
- User/Testing Instructions

The key piece of information is ownership. If you are the author, then most likely you are the owner, or at least the organization you work for is the owner. It becomes more difficult when the component is a mixture of your work and others. Ownership may not be easily established and may take a legal opinion. It is even more difficult to establish ownership when you are supplying a component from another source (referred to as 3rd party supply). This type of supply is discouraged.

Component Coding Guidelines

Detailed guidelines for coding Ada in a reusable style are contained in the *STARS Reusability Guidelines* [IBMCode]. These guidelines were originally published in the *Consolidated Reusability Guidelines* [IBM380]. Essentially, the guidelines are the same as the guidelines which were collectively established by the STARS prime contractors in the STARS Q-Increment. Some of the original guidelines were modified for clarity and depth of definition. A few were eliminated due to experiences and comments since they were published. The guidelines have also been organized to match the metrics produced by AdaMAT.

The guideline document contains a checklist of the guidelines which is very helpful during code development and review. This same checklist is applied to the component code during component filtering by a Topic Specialist.

Coding style is distinct from code format. Code format effects readability but not reusability to any great degree. Code format can be easily handled by formatting tools. Coding style addresses the language elements you use to express the program design. It has an impact on reusability, especially with regard to maintenance and program understanding. But even coding style is not as important to reusability as compilation correctness and documentation completeness.

Col. Whitaker established the STARS program philosophy on coding style with the following note:

> STARS does not wish to impose an excessive or restrictive style on the programmer. A sensible attention to readability and portability should be sufficient guide.
>
> STARS style recommendations have to be consistent with the widest variety of operations, including the thousands of individual shops which may have local ideas, restrictions, and formats enforced by local methods. STARS, therefore, is not restrictive without compelling reason, especially in those areas where it possible to machine restructure the code to any desired style.
>
> STARS sets no specific formatting requirements, as a matter of principle. The philosophy is that one might expect to receive code from various organizations with different ways of doing things. The government will pretty-print to Ada LRM style. The only style limitation is that one should not attempt to encode information (e.g., into the case of identifiers, since Ada is case insensitive), or use other non-Ada conventions. The government should be able to restructure and extract code information that is processable by an Ada compiler.
>
> STARS is trying to develop a software technology to be used by the DoD, not just to control a small group of in-house programmers. The government should not over-specify those things it can easily adapt. Style guidelines that impose more rigid formatting rules are officious pedantry, but very common. Conventions like "_TYPE" may be used by some groups; STARS would not interfere, nor would it attempt to impose them on anyone else.

Arbitrary restrictions to the full capability of Ada (such as unnecessary injunction against "use") are inappropriate. Each local shop may, for its own reasons, add additional restrictions, although STARS would recommend against anything that would limit the expressiveness of Ada. Examples of oppressive limitations include: no "function" in Ada PDL so it can be mapped to COBOL; no "if" nested under an "if," because a tool was derived for a language without "elsif"; forbidding the use of "use," thereby denying much of Ada overloading; forbidding the "while" construct in favor of loops with exit.

STARS is experimenting with using SGML encoding for program prologue information so it can be computer processed. This documentation technique is considered separable from "Ada style," and would be the subject of other guidelines.

Repository Component Supplier Procedure

The repository information model was modified during the R-extended development period. While the operational repository was being adapted this new model, component acquisition had to proceed. The solution has to save components as tagged SGML files, and write a supply tool that would load the data base, using the SGML file as input, as soon as the data base had been adapted to the new model. This is an interim solution until the development of Release 2.5 of the Repository, with its new interactive, integrated Supply Process, becomes operational during September of 1990.

The STARS repository supply tool works as follows. The supplier generates an SGML-tagged text file defined by the Loadstar DTD (See Figure B-2). This file is referred to as an asset file. The supplier provides the text associated with the tags in the asset file. For the "id" tags (termid, orgnzid, personid, cdrlid, assetid), the supplier provides the ID value of an object that already exists in the STARS repository. A complete list of these objects, and their associated ID values, can be obtained from the STARS repository librarian. For the other tags, the supplier provides information derived from the asset. The supplier then constructs a load file, containing a list of the names of the asset files to be loaded, and informs the STARS repository librarian of its location. The librarian passes the load file as input to the STARS repository supply tool. The supply tool reads the names of the asset files from the load file. This tool then reads each asset file and inserts the corresponding asset into the STARS repository. Once all the assets have been inserted, the supply tool generates a log file. The log file is an SGML-tagged text file defined by the Loadlog DTD.

Figure B-2 defines the loadstar DTD and figure B-3 defines the loadlog DTD.

Appendix C. Evaluating Components

Repository Content Management

The Repository content is managed with three levels of component quality: organized, filtered, certified.

Organized Level (Entry to Repository)

Requirements

To be admitted to the organized level of the repository, a component must meet certain minimal data requirements and the information must be entered in the supply form. The component must be classified within the standard taxonomy used by the repository.

Procedures

The repository librarian supervises the admission of a component into the repository.

Filtered Level

Requirements

A component attains the filtered level of the repository through the filtering process. The component must meet more stringent data requirements and undergo analysis. The component must be evaluated for reusability and portability. They must be assessed for their conformance to the coding guidelines prescribed for reusable components.

Procedures

The librarian supervises the filtering process. Topic specialist are brought in to evaluate and analyze components. Specific attributes of the component are determined and recorded. Automated tools, such as AdaMAT, are used where appropriate.

Certified Level

Requirements

The certified level is attained when a component is determined to be 'correct'. Currently, the technology in this area is not refined and the requirements are not established.

Procedures

A component is certified through a process that includes analysis and testing. The component is placed under strict configuration management following the certification.

Component Evaluation Filters

Filtering of components in the repository is the process of reviewing components (and their parts) in order to establish a defined level of understanding about the component. For each filtering process a component attribute or report is recorded and placed in the repository.

The current filters are described in the following paragraphs in the order of their application.

1. COMPILATION

   To pass the compilation filter all the Ada parts of the component must be compiled by the VAX Ada compiler. The success of this effort is recorded in a component attribute. A list of 'withs' outside of the component and a compilation order list must have been supplied when the component was 'organized' or this filter will fail.

2. DOCUMENTATION

   To pass the documentation filter, the existing documentation parts are read and evaluated by a topic specialist. A report on the evaluation of the documentation may be written and added to the component. The general filter is a pass or fail answer to the question, "Is the document sufficient to support component reuse?"

   The documentation will be subjectively evaluated for consistency, clarity, completeness, and correctness. The documentation may be checked for spelling and grammar using automated checkers. The automated checkers also permit metric evaluations, such as 'grade level' and 'word counts'.

   If 'tagged' or pre-formatted documentation exist, it will be passed against the appropriate processors. For example, SGML tagged documentation will be passed through an SGML processor, and PostScript output will be sent to a PostScript printer.

3. METRICS

   AdaMAT will be used to establish the metrics of a coding component. A report will be added to the component that contains a roll-up of all the Ada parts. Reports may be prepared for each Ada part. While there is no specific criteria, components that display very low values may be considered for removal from the Repository.

4. REUSEABILITY

   This filter requires the application of the current STARS reusability coding guidelines fIIBMCode" to the component. It is a subjective review which can be augmented with references to the AdaMAT metrics. The report is for the whole component.

   There is no specific level, rather a pass or fail is recorded for each class of reusability guidelines. The coding issues are subordinate to the overall issues of basic acceptability.

5. TESTING

   To exercise this filter, the component must be executed using the testing parts in the component. If the component has no testing parts, then it can't pass this filter. In some cases it may be reasonable to create testing parts, especially for good components where the supplier didn't contribute any testing parts. Ideally, the report is a test report with "successful" marked against most cases.

6. SECURITY

   This filter involves manual inspection for security problems. Each component part should be viewed on-line and in printed form. All parts are currently maintained in ASCII form. The Repository doesn't support reuse of executables. The review must be noted in the component attribute.

7. CORRECTNESS

The notion of correctness permits a component to be classified as certified. There are several approaches to this subject, which are being studied under STARS tasks. Currently this filter is not available.

Topic Specialist Procedures

The skill and knowledge of the topic specialist is critical to the filtering process. The specialist must judge components according to evaluation criteria in the filtering procedures. They are not mechanical, but rather subjective. The main emphasis is on reviewing the component for its potential reuse.

This process is weakly defined at this stage of the Repository iterative life cycle. As the system matures, it is expected this area will receive more attention and process feedback.

An automation of some of the filtering process is captured in the Repository Gatekeeper facility. This facility, while only an experimental prototype, does provide some useful notions of filtering. It may be used by topic specialists or submitters to coordinate a component evaluation. The facility is invoked by entering "GATEKEEP" at the VAX command prompt. Figure C-1 is a screen print from the execution of the Gatekeeper.

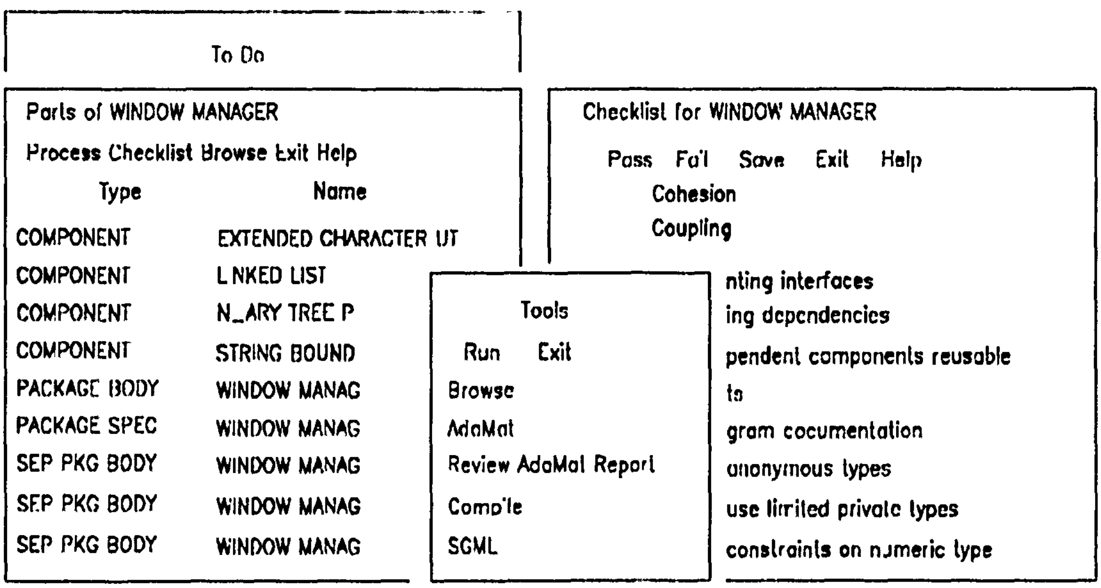


Figure C-1. Gatekeeper Facility

Appendix D. IBM STARS Repository System

Repository Design

The design of the Repository followed the STARS basic integration model and functional interface standards as defined in the Q-increment fIIIBM70". A portability layer was defined that permits migration of the system to other Ada environments. Concessions were made to non-portable development involving the file system, the database manager SQL interface, and the database screen interface aid. The following diagram expresses the general architecture.

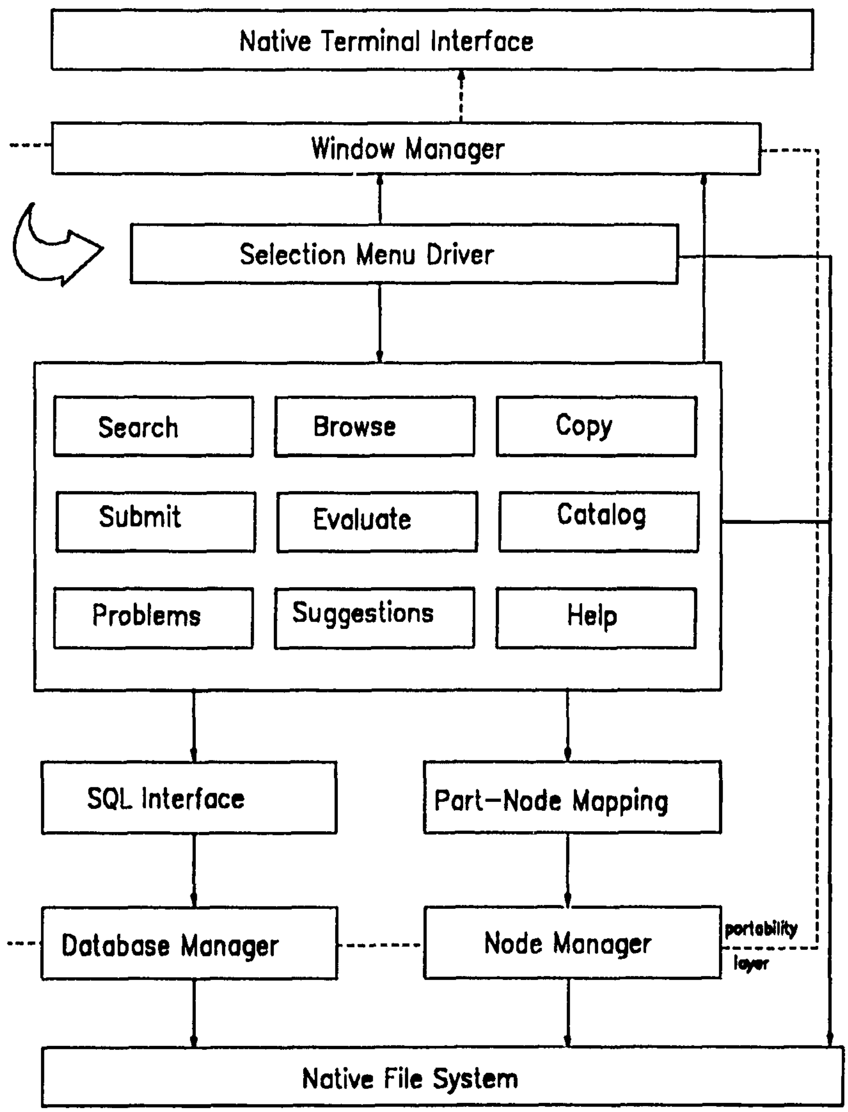


Figure D-1. Repository Basic Architecture

Repository System Facilities

The IBM STARS Repository runs on the following system:

- Hardware System
  - DEC microVAX 3600 with 32 megabytes of memory
  - 1.2 gigabytes of disk storage
  - 18 dial-in modems (16 at 2400 bps, 2 at 9600 bps)
    - 16 at 2400 bps, MNP Level 5 error correction/compression
    - 2 at 9600 bps, MNP Level 5 error correction/compression
  - high speed link to Gaithersburg, 9.2 kbps
- Software System
  - operating system: VAX/VMS Version 4.7
  - database management system: Oracle Version 5.1
  - electronic mail: VMS Mail Version 4.0
  - electronic conferencing: ANU NEWS Version 5.8
  - file transfer: Kermit-32 Version 3.2.077
  - Ada compiler: DEC Ada for VMS Version 1.4
  - Ada analysis: DRC AdaMAT Release 2.0

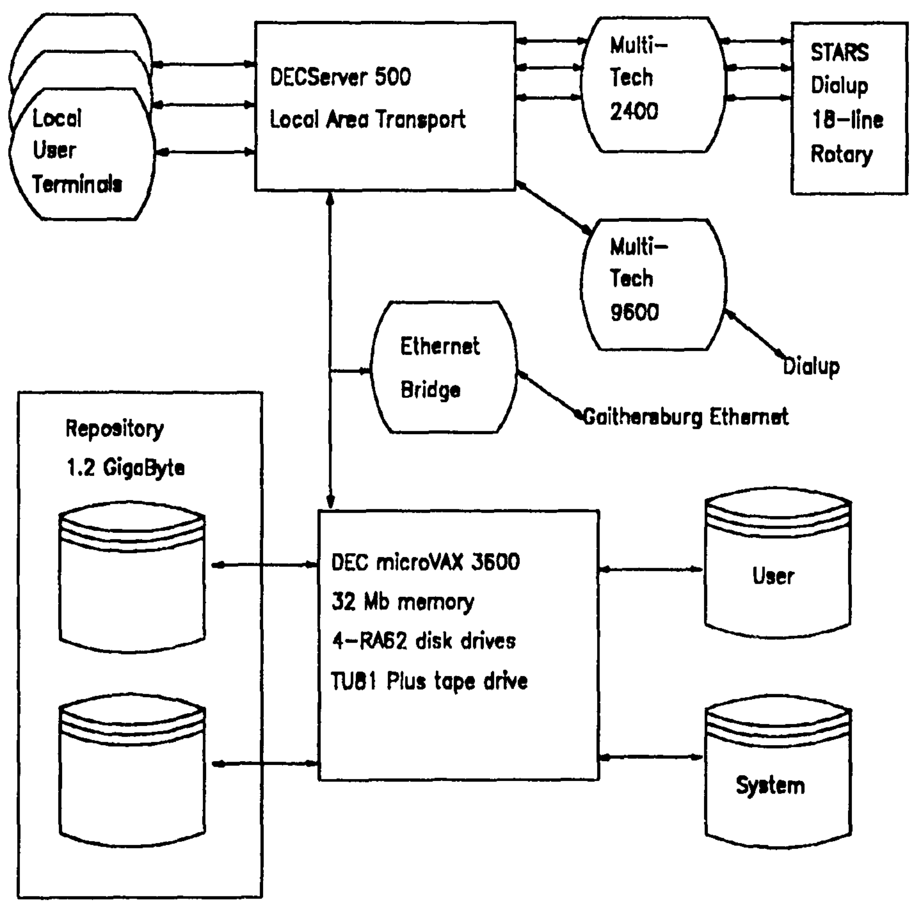


Figure D-2. Repository System Configuration

Repository Capabilities

The following is a list of the capabilities of the IBM STARS Repository [IBM110]:

- Support for Reuse Library
  - component search/select
    - by hierarchy class
    - by facet
    - by attribute
  - component browse (examine)
  - component copy (extract)
  - component tracking
  - component subscription
  - component problem reporting
  - component submit
  - component catalog generation
  - component catalog browse
- Support for Software Engineering
  - Ada compilation
  - Ada metric analysis
  - SGML document preparation
- Support for General Information
  - directory search
  - file browse
  - electronic mail
  - electronic conferencing
  - file transfer (upload/download)
  - people search (scan)
  - system problem reporting
  - usage statistics browse

Repository Database

Information Model

The heart of the Repository is the database and the information model that it implements. The IBM STARS Repository information model is based on a 'component' and its 'parts'. The model is currently being revised, but the following diagram of a recent model presents the basic concepts.

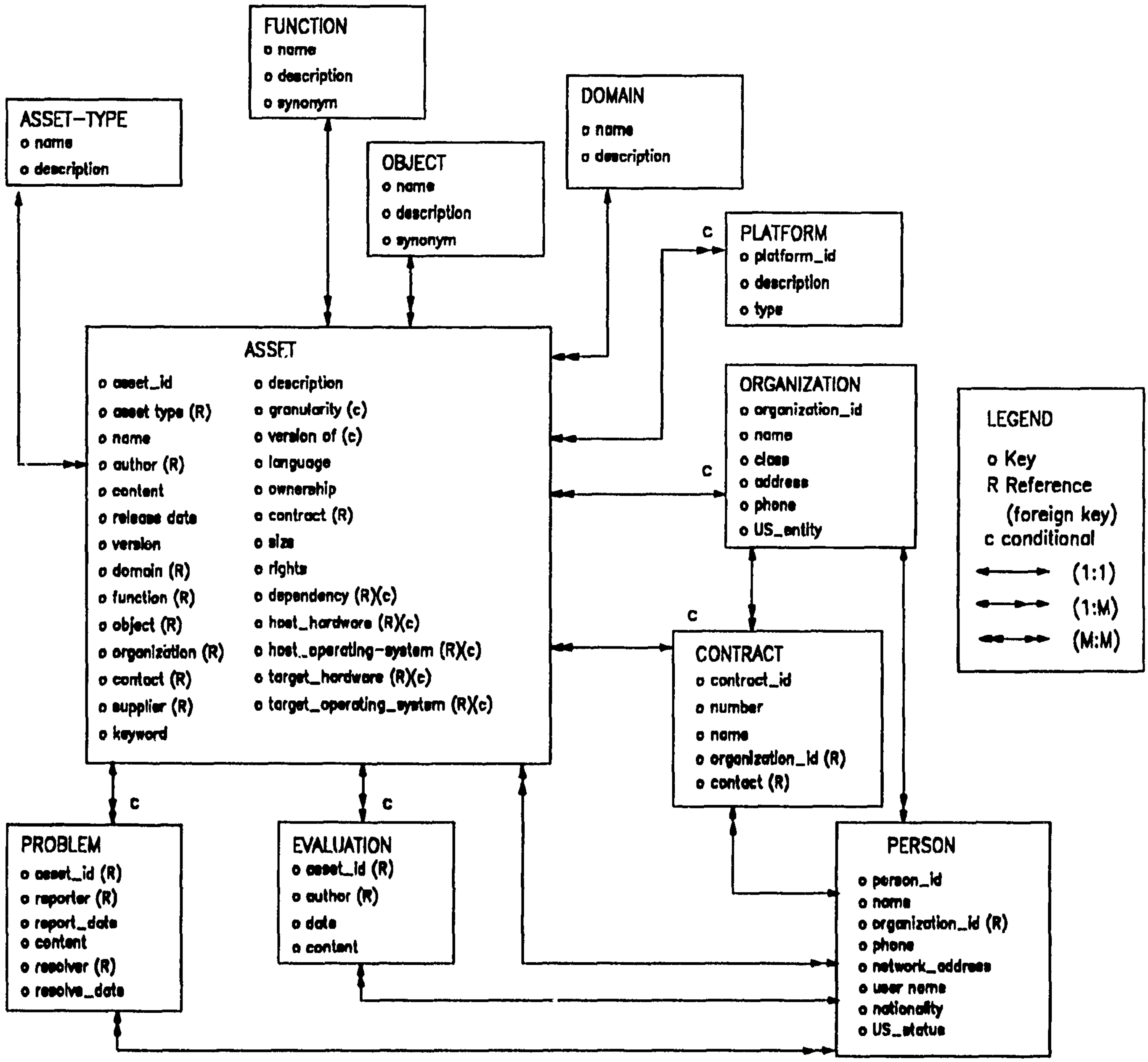


Figure D-3. Repository Information Model

Database Table Definitions

In using and administrating the repository, you will frequently need to know the permitted values (or vocabulary) for particular component attributes. They are presented in this document for users to develop an understanding of the type and depth of the table content. These values are stored in the Repository database and are modified over time as appropriate. All processes in the Repository system that use these values, retrieve them from the current database. To view accurate actual values, you must refer to the current database.

Part Types

The following is a list of PART TYPES available in the Repository.

DESCRIPTION	Description of the component
HISTORY	History of component including author, company, address, etc
DESIGN	Design notes for component
REQUIREMENTS	List of modules or building blocks necessary to run
UNITS	Package name of component
DOSFILE	Name of DOS file in which the component's code is stored
RIGHTS	Copyrights and release agreements
SPEC	Ada specification for the component
BODY	Ada body of code for the component
TEXT	File containing textual characters, intended to be viewed
SGML	File intended to be viewed via some SGML processor

Facets

The following is a list of FACETs and FACET TERMs in the Repository "IBM1580".

FACET	FACET TERM	ALIASES
FUNCTION	ANALYZE	EVALUATE
	CHECK	VERIFY
	DECODE	TRANSLATE
	EXAMINE	DISPLAY,BROWSE
	FORMAT	TRANSFORM
	MANAGE	
	MANIPULATE	APPEND,EXTRACT
	MODIFY	
	USE	
OBJECT	DATES	CALENDAR
	EXTENDED_CHARS	
	FILES	SOURCE,TEXT
	LISTS	
	STACKS	
	STRINGS	
	TREES	GRAPHS
	WINDOWS	
MEDIUM	ARRAYS	
	FILES	
	RECORDS	
SYSTEM_TYPE	FILE_HANDLER	
	SCHEDULER	
	USER_INTERFACE	
FUNCTIONAL_AREA	DOS	
	SWE	
SETTING	SWE	

As the repository grows and the contents become more diverse, the Facets, Terms, and Aliases will be re-examined to be consistent in the meaning of terms and the scattering effect of limiting the number of aliases used. If the composite terms begin to make too large a selection-set then remove one or more aliases and select one for a new primary term. Components being added can then be tied to the new, primary term and all existing components that better fit the new term can be linked to the new term.

Component Classes

The following is the high level list of the hierarchical component CLASS values in the Repository. Each class several several levels of subclasses. This is the basic taxonomic structure of the Repository content.

```
0   STARS PRIMES - INTERCHANGE
1   STARS Enhanced Repository Taxonomy
2   BOEING DELIVERABLES
3   IBM DELIVERABLES
4   UNISYS DELIVERABLES
5   STARS FOUNDATIONS (by Contractor)
6   NOSC_WIS (by Contractor)
7   SIMTEL20
8   CAMP (Common Ada Missle Packages)
9   SDME (S/W Dev. & Maint. Env.)
10  Etc.
```

Development Guidance

The Repository design and implementation followed several general sets of guidance. The development followed the IBM STARS Quality Assurance and Configuration Management Plan [IBM1320]. To review the code work products refer to the in the code delivery of the STARS IR40 task [IBM1600].

Typical User

The Repository was designed with the following typical user in mind.

- Little knowledge of the Repository host computer or operating system.
- No knowledge of the Repository database and approaches to database access.
- Reasonable knowledge of Ada.

User Interface

The basic reference for the Repository user interface is the IBM System Application Architecture (SAA) - Common User Access (CUA) [CUA89]. It defines the concepts of windows, title bars, action bars, client areas, buttons, emphasis, and keyboard interactions. The CUA is generally applied to pixal-graphic user interfaces, but we have generalized the to guidance to character graphics because of the limited capabilities of the Repository system and the remote user systems. The user interface also incorporates models and ideas found in character graphic tools commonly used on VMS.

The following are some guidelines used to develop the user selection menus:

- Selection items should be based on the repository capability list.
- All menus should have "Report Problem," "Help," and "Exit."
- Selection items should only exist in one place of the selection menu hierarchy (except for the above items).
- Selection items not implemented should not be on the menu.
- Selection items should be a 'verb' acting on a 'noun' (examples: 'search directory' and 'browse file'.
- Selection items should be upper and lower case, with major words capitalized, and minor words in all lower case.
- The selection character should be the initial character of the 'verb'.
- The selection character should be emphasized and in upper case.
- Submenus should be titled with the higher level menu selection item.
- Help information should be context sensitive.
- Function keys should not be used.

- Common sense and user perception should prevail when these guidelines produce 'funny' situations.

Object Names

The naming of objects (files, tables, commands) has a significant effect on the management, readability, and reusability of a system. For the Repository development, we have established the following naming guidelines:

- Eliminate all file or directory names in code. Try to acquire the names from standard input.
- When it is not possible to eliminate imbedded file names, isolate the names in easy to locate and modify places.
- Do not use dates, version numbers, CDRL numbers, or other project specific names. Name items according to what the item does, and not according to where it is. Names like DEMO2, STARS_PRIME, or REPOS89 inhibit reuse. Names like REPOS_SEARCH and WINDOW_MANAGER are much better.
- Don't use plurals for names (ie use COMPONENT table, not COMPONENTS table; use COLOR type, not COLORS type).

Index